ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ ΚΑΙ ΥΛΙΚΟΥ

Διπλωματική Εργασία με Θέμα:

"Παράλληλη Κατηγοριοποίηση Κεφαλίδας Πακέτων με Χρήση Γραφικού Επεξεργαστή"

Μακρής Ιωάννης

Εξεταστική Επιτροπή:
Καθηγητής Πνευματικάτος Διονύσιος (επιβλέπων)
Καθηγητής Δόλλας Απόστολος
Επίκ. Καθηγητής Παπαευσταθίου Ιωάννης

Χανιά,2010

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέπων Καθηγητή Κ. Διονύσιο Πνευματικάτο, για την ανάθεση του θέματος και την πολύτιμη και μας δημιουργική συνεργασία.

Επίσης, θα ήθελα να ευχαριστήσω τον Επίκουρο Καθηγητή Κ. Ιωάννη Παπαευσταθίου και τον Καθηγητή Κ. Απόστολο Δόλλα για τη συμμετοχή τους ως μέλη της εξεταστικής επιτροπής.

Τις ευχαριστίες μου θα ήθελα να εκφράσω και στον υπεύθυνο του εργαστηρίου Μικροεπεξεργαστών και Υλικού κ. Μάρκο Κιμιωνή, καθώς και σε όλα τα μέλη του εργαστηρίου για την βοήθεια που μου παρείχαν και για το άψογο κλίμα που υπήρχε στις μεταξύ μας σχέσεις.

Τέλος, θα ήθελα να ευχαριστήσω ιδιαίτερα και να αφιερώσω τη συγκεκριμένη εργασία στην οικογένειά μου και στην αγαπημένη μου φίλη Χριστίνα, που μου συμπαραστέκονται πάντα και με αγαπούν.

Περίληψη

Μια σημαντική εργασία στα συστήματα ανίχνευσης εισβολών σε δίκτυα, είναι η κατηγοριοποίηση επικεφαλίδας των διερχόμενων πακέτων. Με την αύξηση των ρυθμών μετάδοσης δεδομένων στα σύγχρονα δίκτυα και ειδικότερα στο διαδίκτυο, η κατηγοριοποίηση των πακέτων πρέπει να γίνεται όσο το δυνατόν πιο γρήγορα. Διάφορες τεχνικές έχουν προταθεί που στη πλειοψηφία τους είναι προσανατολισμένες στο υλικό.

Τα τελευταία χρόνια, η ραγδαία ανάπτυξη των πολυπύρηνων καρτών γραφικών έχει ανοίξει νέους ορίζοντες στην παράλληλη επεξεργασία. Οι περισσότερες κάρτες γραφικών, πλέον έχουν τη δυνατότητα να εκτελούν και κώδικα γενικής χρήσης εκτός από εφαρμογές γραφικών.

Στη παρούσα εργασία υλοποιήθηκε η λειτουργία της κατηγοριοποίησης κεφαλίδων πακέτων που χρησιμοποιεί το Snort, ένα διαδεδομένο σύστημα ανίχνευσης εισβολών, σε μια κάρτα γραφικών της NVIDIA που υποστηρίζει το μοντέλο της CUDA.

Από τα πειραματικά αποτελέσματα προέκυψε ότι μπορεί να επιτευχθεί επιτάχυνση έως και 100x, έναντι της κατηγοριοποίησης που εκτελεί το Snort.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Γενικά

Τα τελευταία χρόνια παρατηρείται μια αλματώδης ανάπτυξη του διαδικτύου και των υπηρεσιών που προσφέρει. Όλο και περισσότεροι άνθρωποι χρησιμοποιούν το διαδίκτυο πλέον καθημερινά, οι εφαρμογές που προσφέρει αυξάνονται συνεχώς, όπως επίσης οι απαιτήσεις των χρηστών για μεγαλύτερες ταχύτητες, αλλά και ο όγκος των δεδομένων που διακινείται διαμέσω του διαδικτύου. Παράλληλα, αυξάνεται και η ανάγκη για προστασία από μη εξουσιοδοτημένη και κακόβουλη χρήση. Γι' αυτό το λόγο έχουν αναπτυχθεί συστήματα προστασίας, όπως τα τοίχη προστασίας (firewalls) και τα συστήματα ανίχνευσης εισβολών σε δίκτυα (Network Intrusion Detection Systems, NIDS).

Τα firewalls είναι συστήματα λογισμικού ή/και υλικού, τα οποία ελέγχουν τις επικεφαλίδες (headers) των διερχόμενων πακέτων και επιτρέπουν ή απαγορεύουν την πρόσβαση σε συγκεκριμένους χρήστες. Η πιο συνηθισμένη χρήση τους είναι η προστασία του δίκτυου ενός οργανισμού (intranet) από υπολογιστές που δεν ανήκουν στο δίκτυο. Κάθε εισερχόμενο ή εξερχόμενο μήνυμα περνά από το firewall και εμποδίζεται ή επιτρέπεται η διέλευσή του ανάλογα με τα κριτήρια προστασίας που έχουν οριστεί.

Τα συστήματα ανίχνευσης εισβολών σε δίκτυα είναι μεταγενέστερα χρονικά και αποτελούν επέκταση των firewalls. Εκτός από τον έλεγχο της επικεφαλίδας των διερχόμενων πακέτων, τα NIDS συστήματα εκτελούν και έλεγχο των περιεχομένων (payload) των πακέτων για ύπαρξη συγκεκριμένων συμβολοσειρών. Οι συμβολοσειρές αυτές είναι γνωστές ως υπογραφές ή κανόνες. Λόγω της πολυπλοκότητάς τους υλοποιούνται μόνο σε λογισμικό και καταναλώνουν αρκετούς υπολογιστικούς πόρους. Συνήθως, χρησιμοποιούνται για τον εντοπισμό ενός αριθμού από επιθέσεις, όπως υπερχείλιση ενταμιευτή (buffer overflow), ανίχνευση πόρτας (port scans), μπλοκάρισμα μηνύματος

εξυπηρετητή (Server Message Block, SMB) και αποτύπωμα λειτουργικού (OS fingerprint).

Το Snort είναι ένα ευρέως χρησιμοποιούμενο, ανοιχτού κώδικα σύστημα ανίχνευσης εισβολών σε δίκτυα. Ένα σημαντικό κομμάτι του Snort, αλλά και των περισσότερων συστημάτων NIDS, είναι η κατηγοριοποίηση των επικεφαλίδων των διερχόμενων πακέτων. Η κατηγοριοποίηση γίνεται με βάση ένα σύνολο κανόνων που παρέχει το Snort. Η κύρια ιδέα είναι ότι οι κανόνες χωρίζουν τα πακέτα σε κατηγορίες, σύμφωνα με τις παραμέτρους στη κεφαλίδες των πακέτων. Όταν φτάσει ένα πακέτο από το δίκτυο, το Snort βρίσκει σε ποια κατηγορία ανήκει το πακέτο και στη συνέχεια αν βρεθεί κάποια κατηγορία, τα περιεχόμενα του πακέτου ελέγχονται για κάποια ή κάποιες συμβολοσειρές. Αν δεν βρεθεί κατηγορία, το πακέτο δεν θεωρείται ύποπτο και δεν ελέγχεται περαιτέρω. Η κατηγοριοποίηση των επικεφαλίδων των πακέτων είναι μια από τις πρώτες ενέργειες του Snort και μια από τις πιο κρίσιμες τόσο όσον αφορά στην ασφάλεια, όσο και στην ταχύτητα του Snort.

Καθώς τα τελευταία χρόνια οι ταχύτητες αλλά και ο αριθμός των χρηστών των δικτύων αυξάνονται, συνεπάγεται ότι τα συστήματα προστασίας (firewalls και NIDS) πρέπει να λειτουργούν πιο γρήγορα, ούτως ώστε να διασφαλίζεται η μέγιστη δυνατή ασφάλεια αλλά ταυτόχρονα να μην αντιλαμβάνεται μεγάλες καθυστερήσεις ο τελικός χρήστης. Η έλευση των πολυπύρηνων επεξεργαστών και των παράλληλων συστημάτων συνιστά μια τεχνολογική εξέλιξη, η οποία μπορεί να δώσει λύση στο παραπάνω πρόβλημα της επιτάχυνσης των συστημάτων NIDS.

Τα τελευταία χρόνια η ραγδαία ανάπτυξη των καρτών γραφικών και γενικότερα των γραφικών επεξεργαστών (graphics processing units, GPUs), οδηγούμενη από την ανικανοποίητη αγορά για τρισδιάστατα γραφικά πραγματικού χρόνου, έχει ανοίξει νέους ορίζοντες στην παράλληλη επεξεργασία (parallel computing). Πλέον, οι περισσότερες κάρτες γραφικών αποτελούνται από εκατοντάδες επεξεργαστές, οι οποίοι μπορούν να λειτουργούν ταυτόχρονα ή παράλληλα. Πέραν τούτου, πολλές GPU έχουν την δυνατότητα να εκτελέσουν εκτός από εφαρμογές γραφικών και κώδικα γενικής χρήσης. Το μοντέλο Compute Unified Device Architecture (CUDA) είναι ένα προγραμματιστικό μοντέλο που αναπτύχθηκε από την Nvidia και παρέχει στους προγραμματιστές τη δυνατότητα να αναπτύσσουν κώδικα γενικής χρήσης, ο οποίος θα μπορεί να εκτελείται από τους εκατοντάδες επεξεργαστές της κάρτας γραφικών. Η κύρια ιδέα είναι ότι οι προγραμματιστές μπορούν να γράφουν κώδικα C, ο οποίος θα εκτελείται από τους επεξεργαστές της κάρτας παράλληλα.

1.2 Σχετική ερευνητική εργασία πάνω στο Snort

Το Snort έχει αποτελέσει αντικείμενο πολλών ερευνητικών εργασιών, οι οποίες σαν κύριο στόχο είχαν τη βελτίωση της απόδοσής του. Γενικά, οι εργασίες αυτές προτείνουν λύσεις, οι οποίες είναι προσανατολισμένες:

- στο λογισμικό (software)
- στο υλικό (hardware)
- σε συνδυασμό λογισμικού και υλικού
- στις FPGAs
- στις κάρτες γραφικών

Στη κατηγορία των FPGAs, οι κ. Σουρδής και Πνευματικάτος[20] παρουσίασαν μια επεκτάσιμη, χαμηλής καθυστέρησης αρχιτεκτονική για αναζήτηση συμβολοσειρών στα περιεχόμενα των πακέτων, η οποία χρησιμοποιούσε διοχέτευση (pipelining) για την αντιμετώπιση των προβλημάτων fan-out, αντιστοίχισης και κωδικοποίησης, πετυχαίνοντας συχνότητα λειτουργίας μεγαλύτερες από 340MHz για συσκευές Virtex. Μια βελτιστοποίηση της προηγούμενης αρχιτεκτονικής αποτέλεσε η χρήση προ-κωδικοποιημένων CAMs, πετυχαίνοντας συχνότητα λειτουργίας περίπου 375MHz μειώνοντας ταυτόχρονα το κόστος επιφάνειας σε λιγότερο από 1.1 λογικά κελιά ανά χαρακτήρα[21].

Στη κατηγορία του υλικού, οι κ. Σουρδής, Πνευματικάτος, Wong και Βασιλειάδης[22] παρουσίασαν μια τεχνική hashing, για την πρόσβαση της μνήμης που περιέχει τις συμβολοσειρές προς αναζήτηση. Η τεχνική αυτή πετυχαίνει ρυθμαπόδοση μεταξύ 1.7 και 5.7 Gbps, απαιτώντας μερικά μόνο μπλοκ μνήμης της FPGA και 0.32 έως 0.65 λογικά κελία ανά χαρακτήρα. Μια επέκταση αυτής της εργασίας, αποτελεί η αρχιτεκτονική HashMem[11], η οποία επιτυγχάνει χαμηλού κόστους, ακριβές ταίριασμα των υπογραφών του Snort. Μια επιπλέον επέκταση της HashMem αποτελεί η V-HashMem [13], μια αρχιτεκτονική που χρησιμοποιεί μεταβλητού μήκους πρότυπα σε μια απλή δομή μνήμης, μειώνοντας με αυτό το τρόπο τις δομές μνήμης και τους συγκριτές που απαιτούνται. Επιπλέον, η αρχιτεκτονική αυτή περιλαμβάνει και υποστήριξη για κατηγοριοποίηση κεφαλίδας, ένα χαρακτηριστικό που έλειπε από τις προηγούμενες 2 εργασίες [22, 11]. Μια άλλη τεχνική για τη βελτίωση της απόδοσης του Snort είναι το προφιλτράρισμα πακέτων (Paket Prefiltering) [19]. Η τεχνική αυτή βασίζεται σε υλικό και βοηθά στην μείωση των απαιτήσεων επεξεργασίας της ανίχνευσης εισβολών. Υλοποιώντας το μερίδιο ταιριάσματος κεφαλίδας ενός NIDS συστήματος μαζί με ένα μικρό πρόθεμα ταιριάσματος (σε ένα εύρος 4-8 χαρακτήρων), μπορεί να μειώσει τους περισσότερους από

τους κανόνες και να καθορίσει ένα μικρότερο υποσύνολο από εφαρμόσιμους κανόνες, που μπορούν μετά να ελεγχθούν πιο αποτελεσματικά από ένα μοντέλο πλήρους ταιριάσματος.

Αρκετές εργασίες προσανατολίζονται στην βελτίωση του αλγορίθμου Aho-Corasick, ο οποίος είναι ο αλγόριθμος που χρησιμοποιεί το Snort για την αναζήτηση συμβολοσειρών στα περιεχόμενα του πακέτου. Ο αλγόριθμος Split-AC [7] αποτελεί μια παραλλαγή, η οποία μειώνει τις απαιτήσεις σε μνήμη του κλασσικού Aho-Corasick κατά 28-75%.

Στη κατηγορία του λογισμικού οι Tongaonkar, Vasudevan και Sekar [25] παρουσίασαν μια υψηλού επιπέδου γλώσσα βασισμένη στους κανόνες του Snort, η οποία χρησιμοποιείται για την κατηγοριοποίηση των πακέτων. Η γλώσσα αυτή, μέσω μεταγλωττιστή, μετατρέπεται σε κώδικα C, ο οποίος μπορεί να φορτωθεί στο Snort σαν δυναμική βιβλιοθήκη. Με αυτή τη τεχνική, η κατηγοριοποίηση των πακέτων του Snort επιταχύνεται κατά ένα παράγοντα πέντε.

Το T-Gate[6], από την άλλη, αποτελεί μια λύση με συνδυασμό υλικού και λογισμικού, στο οποίο η κατηγοριοποίηση των πακέτων αλλά και η αναζήτηση των συμβολοσειρών στο σώμα των πακέτων, πραγματοποιείται από απλά φίλτρα κατασκευασμένα στο υλικό. Βρέθηκε ότι η χρήση αυτών των απλών φίλτρων, βελτιώνεται η ρυθμαπόδοση του συστήματος με σχετικά χαμηλό κόστος, διατηρώντας παράλληλα την ευελιξία που προσφέρει το λογισμικό.

Τέλος, τα τελευταία χρόνα, λόγω της ραγδαίας ανάπτυξης των καρτών γραφικών πολλές λύσεις προσανατολίζονται στη χρήση καρτών γραφικών για επιτάχυνση των λειτουργιών του Snort. Το PixelSnort [8], αποτελεί μια μεταφορά του Snort στην κάρτα γραφικών 6800 GT της Nvidia, με χρήση της γλώσσας προγραμματισμού Cg (C for graphics) [1]. Η απόδοση του PixelSnort υπερνικά την απόδοση του συμβατικού Snort κατά 40%. Από την άλλη πλευρά, το Gnort [26], αποτελεί μια υλοποίηση του αλγόριθμου Aho Corasick στη κάρτα γραφικών GeForce 8600GT που υποστηρίζει το μοντέλο της Cuda, το οποίο πετυχαίνει ρυθμοαπόδοση 2.3 Gbit/s.

1.3 Συνεισφορά της εργασίας

Σκοπός της παρούσας εργασίας είναι να ερευνηθεί το κατά πόσο ένα σημαντικό κομμάτι ενός συστήματος NIDS, όπως είναι η κατηγοριοποίηση των επικεφαλίδων των διερχόμενων πακέτων μπορεί να πραγματοποιηθεί με τις δυνατότητες που παρέχει μια σύγχρονη κάρτα γραφικών. Για τις ανάγκες της εργασίας αυτής μελετήθηκε το σύστημα Snort ως σύστημα αναφοράς, για τον έλεγχο ορθότητας και αποδοτικότητας των αποτελεσμάτων της εργασίας.

Η συνεισφορά της παρούσας εργασίας έγκειται στην υλοποίηση της κατηγοριοποίησης κεφαλίδας πακέτων παράλληλα. Ο παραλληλισμός επιτυγχάνεται

με χρήση μιας κάρτας γραφικών που υποστηρίζει το μοντέλο CUDA. Η κάρτα γραφικών που χρησιμοποιήθηκε είναι GeForce 9800 GTX της Nvidia.

Η εργασία αυτή σχετίζεται με το Gnort [26], υπό την έννοια ότι και οι δυο εργασίες πραγματοποιούν υπολογισμούς σε κάρτες γραφικών που υποστηρίζουν το μοντέλο CUDA. Η διαφορά τους έγκειται στο ότι πραγματεύονται διαφορετικά κομμάτια του Snort. Το μεν Gnort στην ουσία ασχολείται με την αναγώριση προτύπων και συμβολοσειρών, το οποίο έπεται της κατηγοριοποίησης της κεφαλίδας των πακέτων. Το τελεταίο είναι το αντικείμενο της παρούσας εργασίας.

1.4 Διάρθρωση της εργασίας

Η διάρθρωση της εργασίας έχει ως εξής:

- Κεφάλαιο 2: Περιγραφή του μοντέλου της CUDA, τόσο σε επίπεδο λογισμικού όσο και σε επίπεδο υλικού
- Κεφάλαιο 3: Περιγραφή του Snort και των βασικών χαρακτηριστικών του
- Κεφάλαιο 4: Περιγραφή της υλοποίησης που προτείνεται και των σταδίων στα οποία ολοκληρώθηκε
- Κεφάλαιο 5: Περιγραφή των πειραματικών μετρήσεων και σχολιασμός των αποτελεσμάτων
- Κεφάλαιο 6: Συμπεράσματα και μελλοντικές επεκτάσεις

Κεφάλαιο 2

Εισαγωγή στο μοντέλο CUDA

Η CUDA (Compute Unified Device Architecture) είναι μια παράλληλη αρχιτεκτονική που αναπτύχθηκε από την NVIDIA. Αποτελεί πλέον την καρδιά όλων GPUs (Graphics Processing Units) της NVIDIA. Οι προγραμματιστές έχουν τη δυνατότητα, να χρησιμοποιήσουν τη CUDA (C με κάποιες προεκτάσεις της NVIDIA) για να προγραμματίσουν αλγόριθμους, που θα εκτελούνται στη GPU. Η αρχιτεκτονική της CUDA υποστηρίζει διάφορες διεπαφές, όπως το OpenGL και το Direct3D, καθώς επίσης μέσω τρίτων έχουν αναπτυχθεί περιτυλίγματα (wrappers) για γλώσσες όπως η Java, η Python, η Fortran και η Matlab.

Στη συνέχεια του κεφαλαίου αυτού θα περιγραφούν οι γενικές αρχές που διέπουν το μοντέλο της CUDA, καθώς επίσης και τι προσφέρει στους προγραμματιστές, αλλά και πως υλοποιείται σε υλικό.

2.1 CUDA: Ένα κλιμακούμενο παράλληλο προγραμματιστικό μοντέλο

Η παράλληλη επεξεργασία σε πολυπύρηνους επεξεργαστές είναι μια από της μεγαλύτερες προκλήσεις, σε επίπεδο λογισμικού, στη βιομηχανία. Στα πρόσφατα χρόνια, έχουν προταθεί διάφορες τεχνολογίες για παράλληλη επεξεργασία, όπως η Multicore Develpment Platform της RapindMind[15], η πολυπύρηνη και CPU/GPU προγραμματιστική λύση της PeakStream[12], ο πολυπύρηνος επεξεργαστής της Tilera[24] και άλλες[17].

Η Compute Unified Device Architecture(CUDA) της Nvidia[9] είναι ένα προγραμματιστικό μοντέλο και ένα περιβάλλον λογισμικού ειδικά σχεδιασμένο να ξεπεράσει αυτή τη πρόκληση διατηρώντας πάντως μια χαμηλή καμπύλη

εκμάθησης για προγραμματιστές, οι οποίοι είναι εξοικειωμένοι με κάποιες ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού, όπως η C.

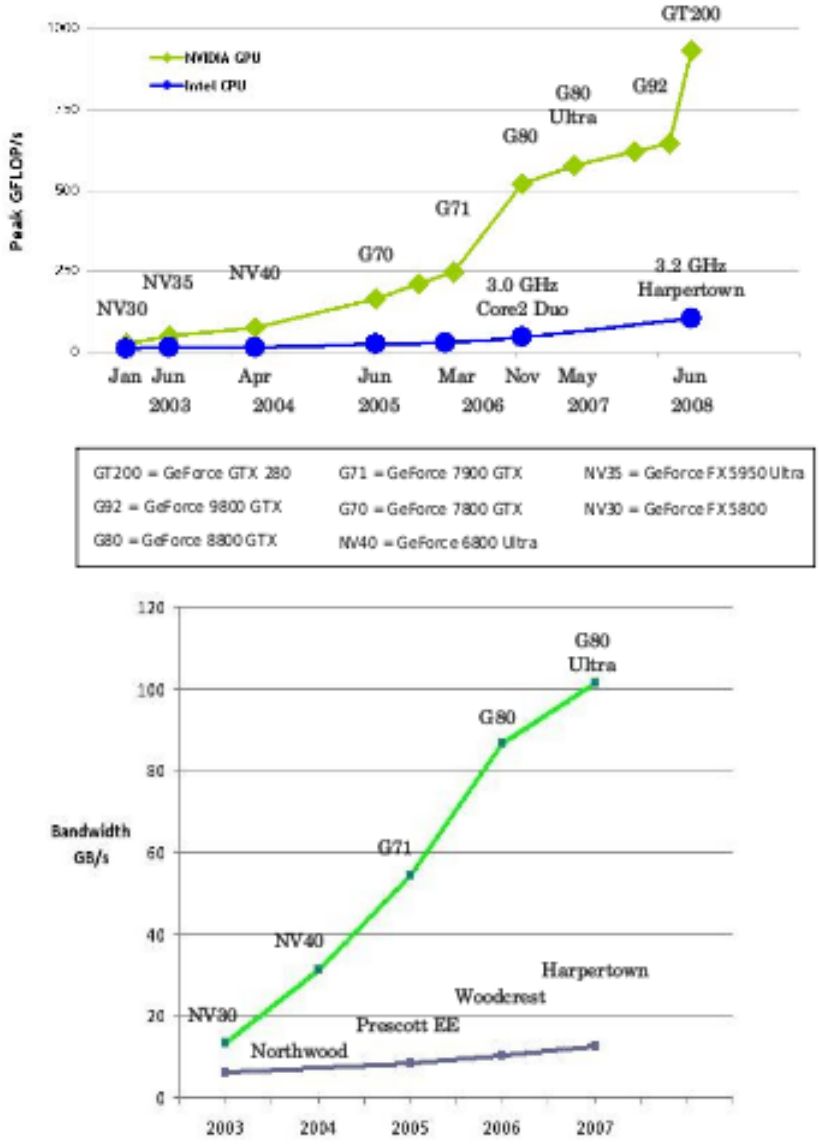
Στο πυρήνα της βρίσκονται τρία αφαιρετικά κλειδιά, τα οποία προσφέρονται στον προγραμματιστή ως ένα σύνολο επεκτάσεων της C. Τα αφαιρετικά αυτά κλειδιά είναι: μια ιεραρχία από νήματα, κοινόχρηστες μνήμες και μπάρες συγχρονισμού.

Αυτές οι αφαιρέσεις παρέχουν έναν καλά ορισμένο παραλληλισμό (fine-grained parallelism) σε επίπεδο δεδομένων αλλά και σε επίπεδο νήματος, καθώς επίσης και ένα πιο αφαιρετικό παραλληλισμό (coarse-grained) σε επίπεδο εργασίας (task). Οδηγούν τον προγραμματιστή να χωρίσει το πρόβλημα σε ανεξάρτητα υποπροβλήματα, τα οποία μπορούν να λυθούν παράλληλα και στη συνέχεια σε μικρότερα κομμάτια, τα οποία μπορούν να λυθούν συνεργατικά παράλληλα. Αυτή η διάσπαση, επιτρέπει στα νήματα να συνεργάζονται μεταξύ τους όταν λύνουν ένα υπο-πρόβλημα και ταυτόχρονα επιτρέπει επεκτασιμότητα (scalability), καθώς κάθε υπο-πρόβλημα μπορεί να χρονοπρογραμματιστεί για λύση σε κάποιον από τους διαθέσιμους επεξεργαστές. Έτσι, ένα μεταγλωττισμένο πρόγραμμα σε CUDA μπορεί να εκτελεστεί σε οποιονδήποτε αριθμό επεξεργαστών και μόνο κατά το χρόνο εκτέλεσης (runtime) το σύστημα πρέπει να γνωρίζει τον αριθμό των επεξεργαστών.

2.2 GPU: Ένας παράλληλος, πολυνηματικός, πολυπύρηνος επεξεργαστής

Οδηγούμενη από την απαίτηση για πραγματικού χρόνου,υψηλής ευκρίνειας τρισδιάστατα γραφικά, η προγραμματιζόμενη GPU εξελίχθηκε σε ένα παράλληλο,πολυνηματικό και πολυπύρηνο επεξεργαστή με φοβερή υπολογιστική ισχύ και παρά πολύ μεγάλο εύρος ζώνης μνήμης (bandwidth), όπως φαίνεται στο σχήμα 2.1 .

Ο λόγος αυτής της διαφοράς μεταξύ του επεξεργαστή και της κάρτας γραφικών είναι ότι η κάρτα γραφικών είναι ειδικά σχεδιασμένη για εφαρμογές, οι οποίες έχουν υψηλές απαιτήσεις σε υπολογισμούς και υπολογισμούς που γίνονται παράλληλα. Γι'αυτό το λόγο, η πλειοψηφία των τρανζίστορ χρησιμοποιείται για επεξεργασία δεδομένων πάρα για έλεγχο ροής (flow control), όπως φαίνεται στο σχήμα 2.2. Πιο συγκεκριμένα, η GPU είναι πιο κατάλληλη για προβλήματα, τα οποία περιέχουν παράλληλους υπολογισμούς δεδομένων, (το ίδιο πρόγραμμα εκτελείται σε πολλά δεδομένα παράλληλα) με υψηλή αριθμιτική ένταση (ο λόγος των αριθμητικών πράξεων προς τις αριθμητικές πράξεις). Επειδή το ίδιο πρόγραμμα εκτελείται για κάθε δεδομένο,δεν υπάρχει καμία απαίτηση για κάποιον πολύπλοκο έλεγχο ροής. Επίσης, επειδή το πρόβλημα



Σχήμα 2.1: Floating-point πράξεις ανά δευτερόλεπτο και bandwidth μνήμης για CPU και GPU

έχει υψηλή αριθμητική ένταση, η καθυστέρηση της μνήμης μπορεί να κρυφτεί από υπολογισμούς μεγάλων συνόλων δεδομένων.

Η παράλληλη επεξεργασία δεδομένων αντιστοιχεί στοιχεία δεδομένων σε παράλληλα νήματα. Πολλές εφαρμογές που επεξεργάζονται μεγάλα σύνολα δεδομένων μπορούν να χρησιμοποιήσουν ενα παράλληλο προγραμματιστικό μο-

Σχήμα 2.2: Η GPU αφιερώνει πιο πολλά τρανζίστορ στην επεξεργασία δεδομένων

ντέλο,για να επιταχύνουν την επεξεργασία τους. Στην τρισδιάστατη απόδοση (3D rendering) μεγάλα σύνολα από pixels αντιστοιχίζονται σε παράλληλα νήματα.Παρόμοια, πολλές εφαρμογές επεξεργασίας εικόνας,όπως για παράδειγμα κωδικοποίσης και αποκωδικοποίησης video και αναγνώρισης προτύπων, αντιστοιχούν μεγάλα μπλοκ δεδομένων σε παράλληλα νήματα. Στην πραγματικότητα, πολλές εφαρμογές και εκτός του πεδίου της επεξεργασίας εικόνας μπορούν να επωφεληθούν από την παράλληλη επεξεργασία των δεδομένων.

Το προγραμματιστικό μοντέλο της CUDA είναι το πλέον κατάλληλο για να δείξει τις παράλληλες ικανότητες των GPUs. Πλέον, όλες οι κάρτες γραφικών τελευταίας γενιάς, που βασίζονται στην αρχιτεκτονική Tesla, υποστηρίζουν το μοντέλο της CUDA και μπορούν να επιταχύνουν τις CUDA εφαρμογές.

2.3 Το προγραμματιστικό μοντέλο της CUDA

Η CUDA επεκτείνει τη C παρέχοντας στους προγραμματιστές τη δυνατότητα να ορίζουν συναρτήσεις C,τα λεγόμενα kernels, τα οποία όταν καλούνται, εκτελούνται παράλληλα N φορες από N διαφορετικά νήματα,σε αντίθεση με τις συνηθισμένες συναρτήσεις C.

Ένα kernel δηλώνεται με το προσδιοριστικό *__global__* και ο αριθμός των νημάτων που θα το εκτελέσουν προσδιορίζεται από τη σύνταξη <<< ... >>>. Κάθε νήμα, που εκτελεί ένα kernel έχει ένα μοναδικό αναγνωριστικό(ID), το οποίο είναι προσβάσιμο μέσα στο κώδικα του kernel, μέσω της μεταβλητής *threadIdx*. Για παράδειγμα, το παρακάτω κομμάτι κώδικα, προσθέτει δυο διανύσματα A και B, μεγέθους N και αποθηκεύει το αποτέλεσμα στο διάνυσμα C:

Η συνάρτηση vecAdd θα εκτελεστεί, με ορίσματα A,B και C, από N νήματα

```
//Kernel definition
__global__ void vecAdd(float* A,float* B,float* C)
{
        int i=threadIdx.x;
        C[i]=A[i]+B[i];
}

int main()
{
        //Kernel invocation
        vecAdd<<<1,N>>>(A,B,C);
}
```

παράλληλα, όπως φαίνεται από τη σύνταξη $<<<1, N>>>$. Το πρώτο όρισμα αλλά και η γενικότερη σύνταξη θα εξηγηθούν παρακάτω.

2.3.1 Οργάνωση των νημάτων

Η μεταβλητή *threadIdx* είναι ένα διάνυσμα τριών διαστάσεων(x,y,z). Έτσι, τα νήματα μπορούν να αναγνωριστούν χρησιμοποιώντας μονοδιάστατους, δισδιάστατους ή τρισδιάστατους αριμοδείκτες(index), σχηματίζοντας αντίστοιχα μονοδιάστατα, δισδιάστατα ή τρισδιάστατα μπλόκ από νήματα (thread block). Υπό αυτή την έννοια, πρόσφερεται ένας φυσικός τρόπος να εφαρμοστούν υπολογισμοί, είτε σε διάνυσματα, είτε σε πίνακες. Για παράγειγμα, το παρακάτω κομμάτι κώδικα, προσθέτει δύο πίνακες A και B, μεγέθους NxN και αποθηκεύει το αποτέλεσμα στο πίνακα C:

Ο αριμοδείκτης ενός νήματος και το ID του σχετίζονται με άμεσο τρόπο: Σε ένα μονοδιάστατο μπλοκ είναι ίδια. Σε ένα δισδιάστατο μπλοκ μεγέθους (Dx,Dy), το ID του νήματος (x,y) είναι (x+y*Dx). Σε ένα τρισδιάστατο μπλοκ μεγέθους (Dx,Dy,Dz),το ID του νήματος (x,y,z) είναι (x+y*Dx+z*DxDy).

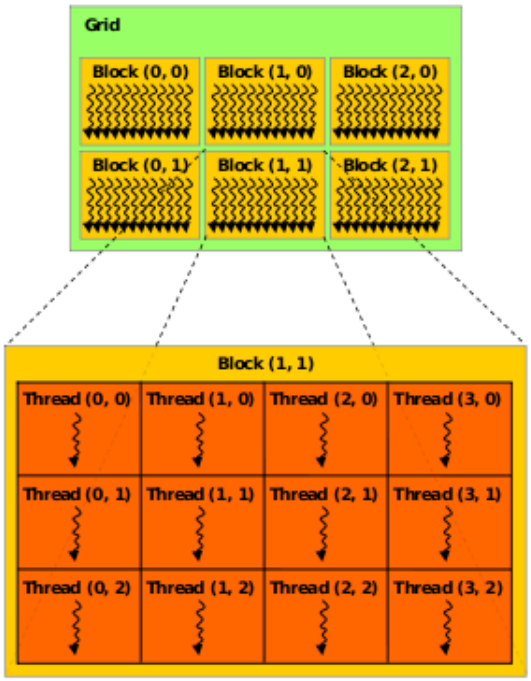
Τα νήματα ενός μπλοκ μπορούν να επικοινωνούν μεταξύ τους και να μοιράζονται δεδομένα διαμέσου της κοινόχρηστης μνήμης, συγχρονίζοντας την εκτέλεσή τους, ώστε να ευθυγραμμίσουν τις προσβάσεις στη μνήμη. Πιο συγκεκριμένα, μπορούν να οριστούν σημεία συγχρονισμού σε ένα kernel, καλώντας τη συνάρτηση __syncthreads(). Η __synthreads() δρα σαν μπάρα, στην οποία τα νήματα πρέπει να περιμένουν μέχρι όλα τα νήματα ενός μπλοκ να φτάσουν, για να τους επιτραπεί να συνεχίσουν.

Ένα kernel μπορεί να εκτελεστεί από πολλά μπλοκ ίδιου μεγέθους, έτσι ώστε, ο συνολικός αριθμός νημάτων να ισούται με το πλήθος των μπλοκ πολλαπλασιασμένο με τον αριθμό των νημάτων ανά μπλοκ. Αυτά τα, ίδιου με-

```
//Kernel definition
__global__ void matAdd(float A[N][N],float B[N][N],float C[N][N])
{
        int i=threadIdx.x;
        int j=threadIdx.y;
        C[i][j]=A[i][j]+B[i][j];
}

int main()
{
        //Kernel invocation
        dim3 dimBlock(N,N);
        matAdd<<<1,dimBlock>>(A,B,C);
}
```

γέθους, μπλοκ οργανώνονται σε ένα πλέγμα(grid) από μπλοκ, όπως φαίνεται στο σχήμα 2.3.



Σχήμα 2.3: Η ιεραρχία των νημάτων

Οι διαστάσεις ενός πλέγματος, προσδιορίζονται από το πρώτο όρισμα της σύνταξης <<< ... >>>. Αντίστοιχα με τα νήματα ενός μπλοκ, τα μπλοκ ενός πλέγματος, μπορούν να αναγνωριστούν από ένα, είτε μονοδιάστατο, είτε δισδιάστατο αριθμοδείκτη, το οποίο είναι προσβάσιμο σε ένα kernel, μέσω της built-in μεταβλητής blockIdx. Η διάσταση ενός μπλοκ είναι προσβάσιμη σε ένα kernel μέσω της built-in μεταβλητής blockDim. Έτσι το προηγούμενο κομμάτι κώδικα μπορεί να γραφεί:

```
//Kernel definition
__global__ void matAdd(float A[N][N],float B[N][N],float C[N][N])
{
        int i=blockIdx.x*blockDim.x+threadIdx.x;
        int j=blockIdx.y*blockDim.y+threadIdx.y;
        if(i<N&&j<N)
                C[i][j]=A[i][j]+B[i][j];
}
int main()
{
        //Kernel invocation
        dim3 dimBlock(16,16);
        dim3 dimGrid((N+dimBlock.x-1)/dimBlock.x ,
                        (N+dimBlock.y-1)/dimBlock.x);
        matAdd<<<dimGrid,dimBlock>>(A,B,C);
}
```

Στο προηγούμενο παράδειγμα κάθε μπλοκ έχει 16x16=256 νήματα και κάθε πλέγμα περιέχει τόσα μπλοκ, ώστε κάθε νήμα να αντιστοιχεί σε κάθε στοιχείο του πίνακα.

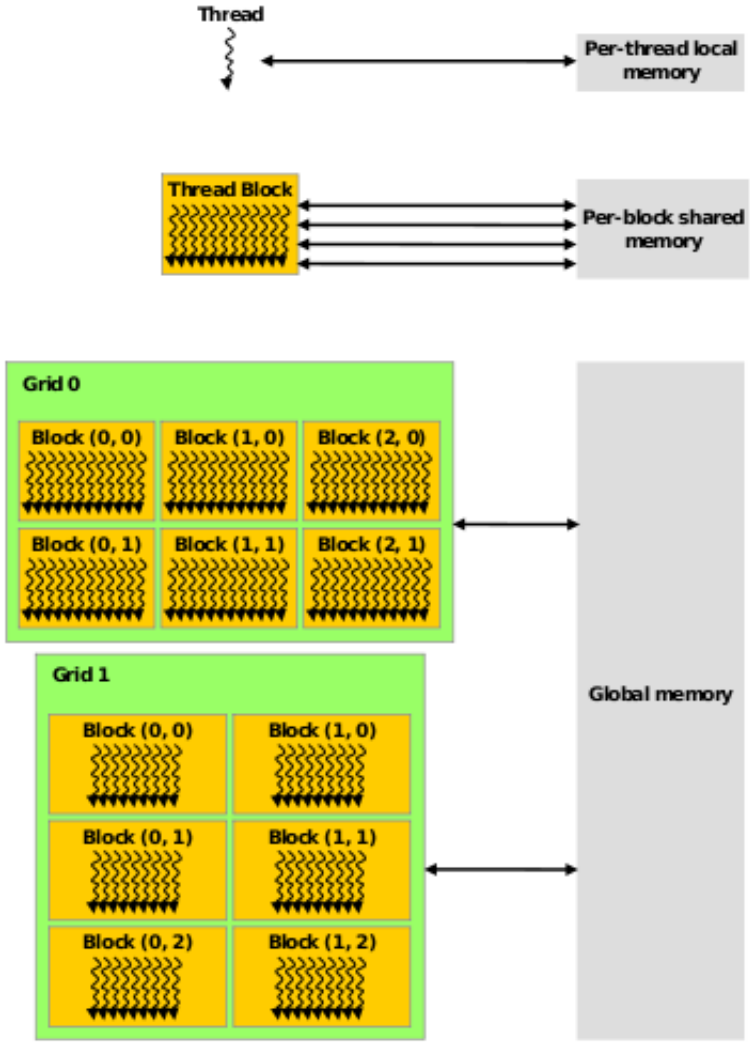
Τα μπλοκ χρειάζεται να εκτελούνται ανεξάρτητα το ένα από το άλλο. Επίσης, πρέπει να είναι δυνατόν να εκτελεστούν παράλληλα. Αυτή η ανάγκη επιτρέπει στα μπλοκ να χρονοπρογραμματίζονται για εκτέλεση σε οποιαδήποτε σειρά σε οποιονδήποτε αριθμό επεξεργαστών.

Το πλήθος των μπλοκ σε ένα πλέγμα καθορίζεται κυρίως από το πλήθος των δεδομένων και όχι από των αριθμό των επεξεργαστών, τον οποίο μπορούν να υπερβαινούν κατά πολύ.

2.3.2 Η ιεραρχία της μνήμης

Τα νήματα στη CUDA μπορούν να προσπελάσουν δεδομένα από πολλούς χώρους μνήμης κατά τη διάρκεια της εκτέλεσής τους, όπως φαίνεται στο

σχήμα 2.4. Κάθε νήμα έχει μια ιδιωτική τοπική μνήμη (local memory). Κάθε μπλοκ έχει μια κοινόχρηστη μνήμη(shared memory), στην οποία έχουν πρόσβαση όλα τα νήματα του μπλοκ. Τέλος, όλα τα νήματα έχουν πρόσβαση στην καθολική μνήμη (global memory).



Σχήμα 2.4: Η ιεραρχία της μνήμης

Υπάρχουν, επίσης, δύο επιπλέον χώροι μνήμης, οι οποίοι είναι μόνο για ανάγνωση: η constant και η texture μνήμη. Η καθολική, η constant και η texture μνήμη είναι βελτιστοποιημένες για διαφορετικές χρήσεις. Η texture μνήμη, επίσης, προσφέρει διαφορετικούς τρόπους διευθυνσιοδότησης, καθώς

επίσης και φιλτράρισμα δεδομένων για κάποιες συγκεκριμένες μορφές δεδομένων.

Η καθολική, constant και texture μνήμη είναι μόνιμες κατά τη διάρκεια της εφαρμογής, δηλαδή οι μεταβλητές που δηλώνονται σε αυτούς τους χώρους μνήμης έχουν χρόνο ζωής το χρόνο ζωής της εφαρμογής. Αντίθετα, η κοινόχρηστη μνήμη έχει χρόνο ζωής το χρόνο ζωής του μπλοκ και η τοπική, το χρόνο ζωής του νήματος.

2.3.3 Host και Device

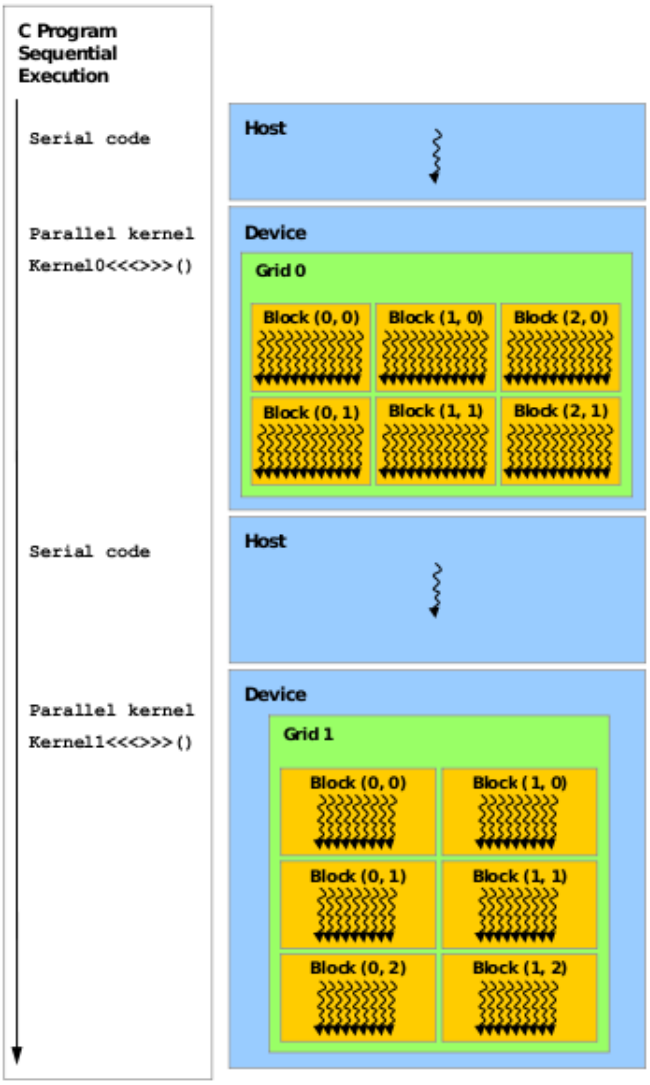
Όπως φαίνεται στο σχήμα 2.5,στη CUDA θεωρείται ότι τα νήματα εκτελούνται σε μια ξεχωριστή φυσική συσκευή, η οποία λειτουργεί σαν συνεπεργαστής στον κύριο επεξεργαστή, το host, ο οποίος εκτελεί το C πρόγραμμα. Αυτή είναι η περίπτωση, για παράδειγμα, στην οποία τα kernel εκτελούνται στη GPU και το υπόλοιπο πρόγραμμα στη CPU. Στη CUDA, επισης, θεωρείται ότι το host και το device έχουν το καθένα τη δική του DRAM, η οποία αναφέρεται ως host μνήμη και device μνήμη, αντίστοιχα.Σαν συνέπεια των παραπάνω, ένα πρόγραμμα διαχειρίζεται την καθολική,την constant και τη texture μνήμη, μέσω κλήσεων στο χρόνο εκτέλεσης της CUDA. Οι κλήσεις αυτές περιλαμβάνουν δέσμευση και αποδέσμευση device μνήμης, καθώς επίσης μεταφορά δεδομένων μεταξύ του host και του device.

2.3.4 Τα επίπεδα του λογισμικού

Όπως φαίνεται στο σχήμα 2.6, υπάρχουν διάφορα επίπεδα λογισμικού: ένας οδηγός για τη συσκευή, ένα API για την εφαρμογή και ο χρόνος εκτέλεσης της, καθώς επίσης και κάποιες βιβλιοθήκες κοινής χρήσης.

2.4 Υλοποίηση της GPU

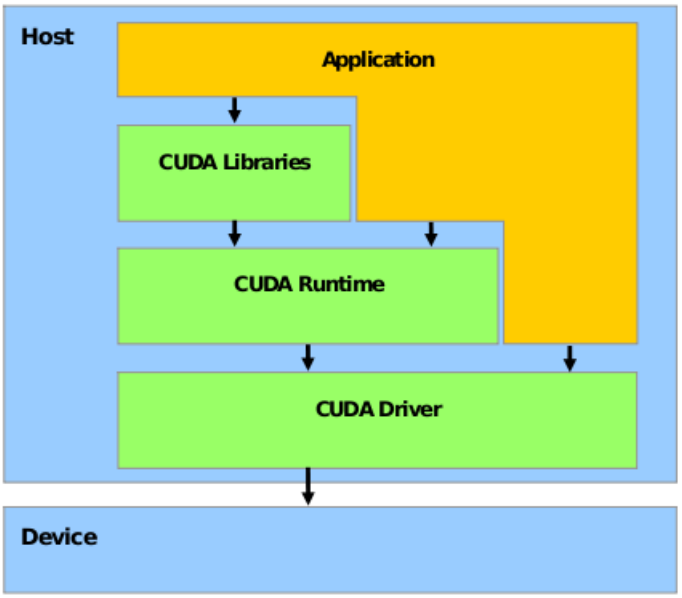
Η αρχιτεκτονική Tesla, της NVIDIA, επεκτείνει τη GPU πέρα από εφαρμογές γραφικών. Οι πολυνηματικοί της επεξεργαστές έχουν μετατραπεί σε μια ενοποιημένη πλατφόρμα για εφαρμογές γραφικών αλλά και εφαρμογές γενικού σκοπού. Τα χαρακτηριστικά της επιτρέπουν το προγραμματισμό των πυρήνων της GPU σε C διαμέσω της CUDA.



Σχήμα 2.5: Ετερογενής προγραμματισμός

2.4.1 Ένα σύνολο SIMT πολυεπεξεργαστών με on-chip κοινόχρηστη μνήμη

Η αρχιτεκτονική Tesla έχει κτιστεί σε μια διάταξη πολυνηματικών πολυεπεξεργαστών ροής (streaming multiprocessors,SMs). Όταν ένα CUDA πρόγραμμα στο host εκτελεί ένα kernel, τα μπλοκ του πλέγματος απαριθμούνται και κατανέμονται στους πολυεπεξεργαστές, οι οποιοί μπορούν να τα εκτελέσουν.Τα νήματα ενός μπλοκ εκτελούνται ταυτόχρονα σε ενα πολυεπεξεργαστή.



Σχήμα 2.6: Τα επίπεδα λογισμικού

Καθώς τερματίζουν τα μπλοκ, καινούργια μπλοκ χρονοπρογραμματίζονται για εκτέλεση στους διαθέσιμους πολυεπξεργαστές.

Ένας πολυεπεξεργαστής αποτελείται από οκτώ scalar επεξεργαστές(SPs), δυο ειδικές υπερβατικές μονάδες(transcendental), μια πολυνηματική μονάδα εντολών και μια κοινόχρηστη μνήμη. Οι πολυεπεξεργαστές δημιουργούν, διαχειρίζονται και εκτελούν τα νήματα στο υλικό, χωρίς καθόλου overhead. Επίσης, υλοποιούν την συνάρτηση __syncthreads() με μια εντολή.

Για να διαχειριστούν εκατοντάδες νήματα, οι πολυεπεξεργαστές χρησιμοποιούν μια νεα αρχιτεκτονική που ονομάζεται SIMT (Single Instruction Multiple Thread). Ο πολυεπεξεργαστής αντιστοιχεί κάθε νήμα σε ένα scalar επεξεργαστή και κάθε νήμα εκτελείται ανεξάρτητα, με τη δική του διεύθυνση εντολής και δικούς του καταχωρητές. Ο πολυεπεξεργαστής δημιουργεί, διαχειρίζεται και χρονοπρογραμματίζει τα νήματα σε ομάδες των 32, τα λεγόμενα warps. Τα νήματα ενός warp ξεκινούν την εκτέλεση από την ίδια διεύθυνση προγράμματος, αλλά είναι ελεύθερα να διακλαδιστούν και να εκτελεστούν ανεξάρτητα.

Όταν ένας πολυεπεξεργαστής έχει ένα ή περισσότερα μπλοκ για εκτέλεση, χωρίζει τα νήματα σε warps, τα οποία χρονοπρογραμματίζονται για εκτέλεση. Ο τρόπος με τον οποίο γίνεται ο διαχωρισμός σε warp είναι συνέχεια ο ίδιος; κάθε warp περιέχει συνεχόμενα νήματα ,με αυξανόμενο προσδιοριστικό νήματος.

Κάθε φορά που γίνεται έκδοση μιας εντολής, η SIMT μονάδα επιλέγει ένα warp, το οποίο είναι έτοιμο για εκτέλεση και εκδίδει την εντολή σε όλα τα ενεργά νήματα αυτού του warp. Το warp εκτελεί μια εντολή τη φορά, έτσι πλήρης αποδοτικότητα επιτυγχάνεται όταν και τα 32 νήματα συμφωνούν στο μονοπάτι εκτέλεσής τους. Αν τα νήματα αποκλίνουν, για παράδειγμα μέσω ενός branch, το warp εκτελεί σειριακά όλα τα μονοπάτια, απεργοποιώντας κάθε φορά τα νήματα που δεν βρίσκονται στο μονοπάτι και όταν όλα τα μονοπάτια τελειώσουν τα νήματα συγκλίνουν στο ίδιο μονοπάτι. Απόκλιση συμβαίνει μόνο μέσα σε ένα warp, διαφορετικά warps εκτελούνται ανεξάρτητα ασχέτως αν εκτελούν ανεξάρτητα κομμάτια κώδικα.

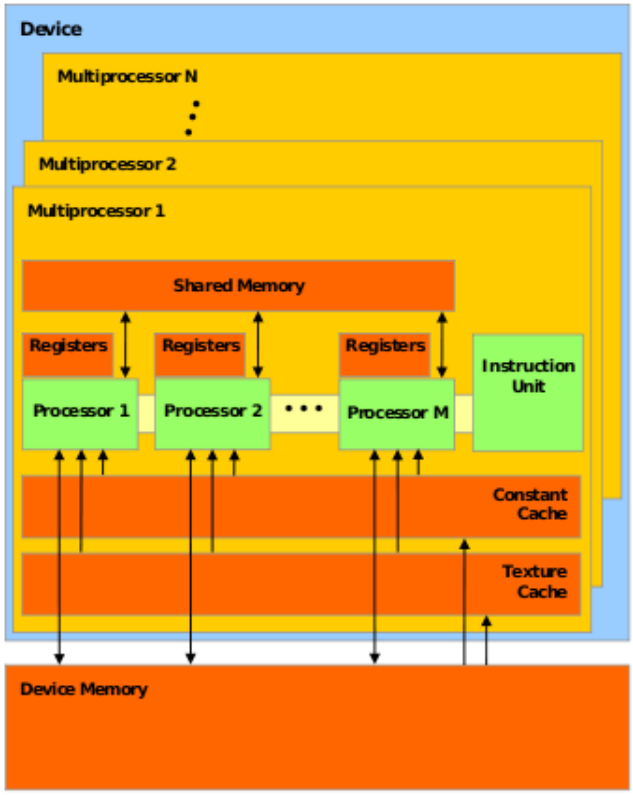
Η SIMT αρχιτεκτονική μοιάζει με την SIMD στο ότι μια εντολή αφορά πολλά στοιχεία επεξεργασίας. Μια βασική διαφορά είναι ότι η SIMD οργάνωση εκθέτει το πλάτος του SIMD στο λογισμικό, ενώ, οι SIMT εντολές προσδιορίζουν την εκτέλεση και την συπεριφορά στα branches ενός νήματος. Σε αντίθεση με τις SIMD μηχανές, οι SIMT επιτρέπουν στους προγραμματιστές να γράφουν παράλληλο κώδικα σε επίπεδο νήματος για ανεξάρτητα νήματα, καθώς επίσης και παράλληλο κώδικα σε επίπεδο δεδομένων για συνεργαζόμενα νήματα. Για λόγους ορθότητας, ένας προγραμματιστής μπορεί να αγνοήσει τη SIMD συμπεριφορά, όμως σημαντικές βελτιώσεις στην απόδοση μπορούν να επιτευχθούν όταν τα νήματα αποκλίνουν σπάνια. Οι vector αρχιτεκτονικές, από την άλλη πλευρά, απαιτούν από το λογισμικό να διαχειρίζεται την απόκλιση.

Όπως φαίνεται στο σχήμα 2.7,κάθε πολυεπεξεργαστής έχει on-chip μνήμη τεσσάρων ειδών:

- Ένα σύνολο τοπικών 32-bit επεξεργαστών ανά επεξεργαστή
- Μια κοινόχρηστη μνήμη η οποία μοιράζεται από όλους τους scalar επεξεργαστές και στην οποία βρίσκεται ο κοινόχρηστος χώρος μνήμης,
- Μια μόνο για ανάγνωση constant cache,η οποία μοιράζεται από όλους τους επεξεργαστές και επιταχύνει ανγνώσεις από τον constant χώρο μνήμης, ο οποίος έιναι μια μόνο για ανάγνωση περιοχή της device μνήμης
- Μια μόνο για ανάγνωση texture cache, η οποία μοιράζεται από όλους τους επεξεργαστές και επιταχύνει αναγνώσεις από τον texture χώρο μνήμης,ο οποίος έιναι μια μόνο για ανάγνωση περιοχή της device μνήμης

Οι τοπικός και καθολικός χώροι μνήμης είναι περιοχές εγγραφής-ανάγνωσης της device μνήμης και δεν είναι εναποθηκευμένοι (cached).

Ο αριθμός των μπλοκ ένας πολυεπεξεργαστής μπορεί να εκτελέσει, εξαρτάται από τον αριθμό των καταχωρητών ανά νήμα και από πόση κοινόχρηστη



Σχήμα 2.7: Το μοντέλο του υλικού

μνήμη ανά μπλοκ απαιτούνται για ένα kernel, αφού οι παραπάνω πόροι του πολυεπεξεργαστή, μοιράζονται στα νήματα ενός πλέγματος. Αν δεν υπάρχουν αρκετοί πόροι για τουλάχιστον ένα μπλοκ, το kernel δε θα εκτελεστεί. Ένας πολυεπξεργαστής μπορεί να εκτελέσει το πολύ οκτώ μπλοκ ταυτόχρονα.

Κεφάλαιο 3

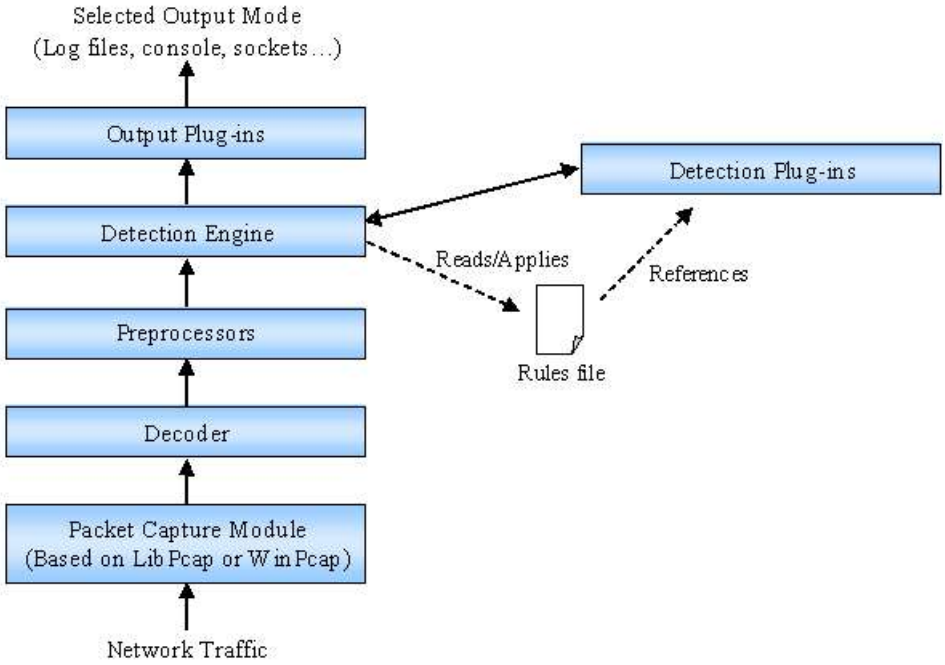
Περιγραφή του Snort

Το Snort[18] είναι ένα από τα πιο δημοφιλή συστήματα ανίχνευσης εισβολών σε δίκτυα (Network Intrusion Detection System,NIDS). Πρόκειται για μια εφαρμογή ανοιχτού κώδικα, η οποία γράφηκε από τον Martin Roesch σε C και τώρα αναπτύσσεται από την Sourcefire και είναι ικανό για ανάλυση κίνησης δικτύου σε πραγματικό χρόνο καθώς επίσης και καταγραφής πακέτων σε IP δίκτυα. Πλέον το Snort έχει καθιερωθει ως το de facto standard στα NIDS συστήματα.

Το Snort μπορεί να εκτελέσει ανάλυση πρωτοκόλλου (protocol analysis), αναζήτηση και ταίριασμα περιεχομένου (content searching/matching) και συνήθως χρησιμοποιείται για τον εντοπισμό και το μπλοκάρισμα μια ποικιλίας επιθέσεων και εισβολών σε δικτύα TCP/IP. Ανάμεσα στις επιθέσεις αυτές είναι, υπερχείλιση ενταμιευτή (buffer overflow), ανίχνευση πόρτας (port scans), μπλοκαρίσματος μηνύματος εξυπηρετητή (Server Message Block, SMB) και αποτυπώματος λειτουργικού (OS fingerprint). Χρησιμοποιεί, μια ευέλικτη και απλή γλώσσα κανόνων για να περιγράψει κίνηση η οποία θεωρείται ύποπτη, καθώς επίσης και μια μηχανή ανίχνευσης (detection engine) η οποία έχει μια τμημακωτή αρχιτεκτονική. Έχει τρείς κύριες χρήσεις: ως απλό packet sniffer για αποτύπωση των πακέτων δικτύου στην οθόνη, ως καταγραφέας πακέτων για την καταγραφή των πακέτων στο δίσκο, ή ως ένα πλήρες NIDS σύστημα για την ανάλυση της κίνησης όπως θα αναφερθεί στη συνέχεια του κεφαλαίου αυτού. Μπορεί να λειτουργίσει σε διάφορα λειτουργικά συστήματα όπως Windows,Unix,Solaris κτλ και είναι διαθέσιμο στην ιστοσελίδα http://www.snort.org. Στη συνέχεια του κεφαλαίου θα περιγραφεί η αρχιτεκτονική του Snort, και η σύνταξη των κανόνων του.

3.1 Η αρχιτεκτονική του Snort

Το Snort όπως φαίνεται στο σχήμα 3.1 το Snort αποτελείται από τα εξής modules:



Σχήμα 3.1: Το block diagram του Snort

Σύλληψης πακέτων (Packet Capture) βασίζεται στη βιβλιοθήκη libcap ή winpcap και σκοπός του είναι να παίρνει τα πακέτα από το δίκτυο

Αποκωδικοποιητής (Decoder) στόχος του είναι να αποθηκεύσει τα πακέτα σε κάποιες δομές δεδομένων του Snort και να αποκωδικοποιήσει το πρωτόκολλο ζεύξης. Στη συνέχεια αποκωδικοποιεί το IP και στη συνέχεια το TCP ή το UDP για να εξάγει χρήσιμη πληροφορία όπως διευθύνσεις IP και πόρτες

Προεπεξεργαστές (Pre-Processors) μπορούν να θεωρηθούν σαν ένα είδος φίλτρου, το οποίο αναγνωρίζει πράγματα τα οποία θα μπορούσαν να εξεταστούν από επόμενα modules, όπως ύποπτες προσπάθεις σύνδεσης σε TCP/UDP πόρτες ή portscan. Η λειτουργία των προεπεξεργαστών είναι να πάρουν πακέτα, τα οποία πιθανόν είναι επικίνδυνα για τη μηχανή ανίχνευσης, και να βρούν γνωστά πρότυπα (patterns)

Αρχεία κανόνων (Rules files) απλά αρχεία κειμένου, με κατάληξη .rules, με συγκεκριμένη σύνταξη που περιγράφουν ύποπτη κίνηση. Ανανεώνονται συχνά όπως οι τα αρχεία ορισμού ιών(virous definitions).

Plugins ανίχνευσης χρησιμοποιούνται για να αναγνωρίζουν πρότυπα όταν υπολογίζεται ένα κανόνας

Μηχανή ανίχνευσης (Detection Engine) το σημαντικότερο ίσως κομμάτι του Snort, αντιστοιχεί, με τη βοήθεια των plugins ανίχνευσης, πακέτα σε κανόνες

Plugins εξόδου επιτρέπουν την μορφοποίηση των προειδοποιήσεων (logs και alerts) και την προσπέλαση τους με διάφορους τρόπους(στην κονσόλα, σε αρχεία , σε βάσεις δεδομένω κτλ)

3.2 Οι κανόνες του Snort

Το Snort χρησιμοποιεί μια απλή, ευέλικτη αλλά και ταυτόχρονα πολύ ισχυρή γλώσσα περιγραφής κανόνων, για να περιγράψει ύποπτη κίνηση. Στις προηγούμενες εκδόσεις του Snort, οι κανόνες γράφονταν σε μια γραμμή. Στις τρέχουσες εκδόσεις, οι κανόνες μπορούν να γράφονται σε πολλές γραμμές, προσθέτοντας ένα \ στο τέλος της γραμμής. Οι κανόνες χωρίζονται σε δυο λογικά κομμάτια, την κεφαλίδα του κανόνα (rule header) και τις ρυθμίσεις (rule option). Η κεφαλίδα περιέχει την ενέργεια του κανόνα, το πρωτόκολλο, τις διευθύνσεις IP του αποστολέα και του παραλήπτη και τις πόρτες αποστολής και λήψεις. Οι ρυθμίσεις περιέχουν μηνύματα προειδοποίησης και πληροφορίες για τα κομμάτια του πακέτου πρέπει να εξεταστούν για αν αποφασιστεί αν θα ακολουθεθεί η δράση του κανόνα. Ένα τυπικό παράδειγμα κανόνα είναι:

```
alert tcp any any -> 192.168.1.0/24 111 \
(content:"|00 01 86 a5|";msg:"mountd access";)
```

Οτιδήποτε μέχρι την πρώτη παρένθεση είναι η κεφαλίδα του κανόνα και το κείμενο μέσα στη παρένθεση είναι οι ρυθμίσεις του κανόνα. Στο κομμάτι των ρυθμίσεων ενός κανόνα, οι λέξεις πριν την άνω και κάτω τελεία, όπως η content και η msg στο πιο πάνω κανόνα, ονομάζονται λέξεις κλειδιά (keywords) και οι λέξεις μετά την άνω και κάτω τελεία είναι η τιμή που παίρνει μια λέξη κλειδί. Να σημειωθεί ότι το κομμάτι των ρυθμίσεων ενός κανόνα είναι προαιρετικό. Χρησιμοποιείται κυρίως για να κάνει πιο συγκεκριμένη την αναζήτηση μέσα σε ένα πακέτο.

Όλα τα στοιχεία ενός κανόνα (κεφαλίδα και ρυθμίσεις) πρέπει να είναι αληθή για να ενεργοποιηθεί η δράση του κανόνα. Έτσι μπορεί να θεωρηθεί ότι τα στοιχεία ενός κανόνα σχηματίζουν μια λογική πρόταση AND. Αντίστοιχα, οι διάφοροι κανόνες, στα αρχεία κανόνων του Snort, σχηματίζουν μια λογική πρόταση OR.

3.3 Οι κεφαλίδες των κανόνων

3.3.1 Η ενέργεια των κανόνων

Η κεφαλίδα ενός κανόνα περιέχει πληροφορίες για το ποιά πακέτα πρέπει να ελεγχθούν καθώς επίσης και το τι πρέπει να κάνει το Snort όταν ανιχνεύσει ένα πακέτο για το οποίο επαληθευέται ο κανόνας. Το πρώτο κομμάτι μιας κεφαλίδας είναι η ενέργεια του κανόνα και καθορίζει τι θα κάνει το Snort όταν ενεργοποιηθεί ένας κανόνας. Το Snort παρέχει 5 προκαθορισμένες ενέργειες, alert,log,pass,activate και dynamic. Πιο συγκεκριμένα:

- alert - παράγει μια προειδοποίηση σύμφωνα με τη μέθοδο προειδοποίησης που έχει επιλεχθεί και στη συνέχεια καταγράφει το πακέτο
- log - καταγράφει το πακέτο
- pass - αγνοεί το πακέτο
- activate - παράγει μια προειδοποιήση και στη συνέχεια ενεργοποιεί ένα δυναμικό κανόνα
- dynamic - παραμένει αδρανές μέχρι να ενεργοποιηθεί ο κανόνας και στη συνέχεια δρά σαν κανόνας καταγραφής

Ο χρήστης μπορεί επίσης να ορίσει τους δικούς του τύπους κανόνων και να τους συσχετίσει με ένα ή περισσότερα plugins εξόδου. Στη συνέχεια, μπορεί να χρησιμοποιήσει αυτούς τους τύπους σαν ενέργειες στους κανόνες του Snort. Για παράδειγμα, μπορεί κάποιος να ορίσει ένα τύπο κανόνα ο οποίος καταγράφει στη syslog και σε μια βάση MySQL:

3.3.2 Τα πρωτόκολλα των κανόνων

Το επόμενο πεδίο σε μια κεφαλίδα είναι το πρωτόκολλο του κανόνα. Προς το παρών, το Snort αναλύει τέσσερα πρωτόκολλα: TCP, UDP, ICMP και IP. Στο μέλλον μπορεί να υποστηρίξει και άλλα όπως: ARP, IGRP, GRE, OSPF και άλλα.

```
ruletype redalert
{
    type alert
    output alert_syslog: LOG_AUTH LOG_ALERT
    output database: log,mysql,
    user=snort dbname=snort host=localhost
}
```

3.3.3 Οι διευθύνσεις IP

Το επόμενο πεδίο της κεφαλίδας είναι η διεύθυνση IP του αποστολέα. Η λέξη κλειδί *any* μπορεί να χρησιμοποιηθεί για να ορίσει οποιαδήποτε διεύθυνση IP. Οι διευθύνσεις σχηματίζονται από μια απλή διεύθυνση IP και ένα μπλοκ CIDR (Classless Inter-Domain Routing). Ένα μπλοκ CIDR υποδηλώνει μια μάσκα δικτύου (netmask) η οποία πρέπει να εφαρμοστεί στη διεύθυνση του κανόνα και στα εισερχόμενα πακέτα που ελέγχονται με αυτόν το κανόνα. Ένα μπλοκ CIDR /24 υποδηλώνει ένα δίκτυο τάξης C, /16 υποδηλώνει ένα δίκτυο τάξης B και /32 υποδηλώνει ένα μια συγκεκριμένη διεύθυνση IP. Για παράδειγμα η διεύθυνση 192.168.1.0/24 αναφέρεται στις διευθύνσεις 192.168.1.1 έως 192.168.1.255.

Στις διευθύνσεις IP μπορεί να εφαρμοστεί και ο τελεστής άρνησης (ο οποίος δηλώνεται με !). Ο τελεστής αυτός καθοδηγεί το Snort, να αντιστοιχίσει οποιαδήποτε διεύθυνση εκτός αυτής στην οποία εφαρμόζεται ο τελεστής. Ένα παράδειγμα φαίνεται στον ακόλουθο κανόνα:

```
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 111 \
         (content:"|00 01 86 a5|";msg:"mountd access";)
```

Επίσης μπορεί να οριστεί και λίστα από διευθύνσεις IP ως εξής:*[IP1,IP2,...,IPn]*. Σημειώνεται ότι ο τελεστής άρνησης μπορεί να εφαρμοστεί και στη λίστα από διευθύνσεις, όπως φαίνεται και στον επόμενο κανόνα:

```
alert tcp ![192.168.1.0/24,10.1.1.0/24] any -> \
       [192.168.1.0/24,10.1.1.0/24] 111 \
         (content:"|00 01 86 a5|";msg:"mountd access";)
```

3.3.4 Οι πόρτες

Το επόμενο πεδίο είναι η πόρτα του κανόνα. Πόρτες μπορούν να δηλωθούν ως εξής:

- με τη λέξη κλειδί *any*, δηλώνοντας οποιαδήποτε πόρτα
- με μια απλή αριθμιτική τιμή, δηλώνοντας την συγκεκριμένη πόρτα
- με ένα διάστημα χρησιμοποιώντας τον τελεστή του διαστήματος : .Για παράδειγμα με τη σύνταξη 20:30 δηλώνουμε όλες τις πόρτες από την 20 μέχρι την 30. Επίσης με τη σύνταξη 20: δηλώνουμε όλες τις πόρτες που είναι μεγαλύτερες από 20, ενώ αντίστοιχα με :20, δηλώνουμε όλες τις πόρτες μικρότερες της 20

Τέλος σε όλες τις παραπάνω περιπτώσεις, εκτός του any, μπορεί να εφαρμοστεί ο τελεστής της άρνησης με την προφανή σημασία.

3.3.5 Ο τελεστής της κατεύθυνσης

Ο τελεστής της κατεύθυνσης ->, δείχνει τη προέλευση ή το προορισμό της κίνησης στην οποία εφαρμόζεται ο κανόνας. Οι διευθύνσεις IP και οι πόρτες αριστερά του τελεστή αναφέρονται στην προέλευση, ενώ αυτές δεξιά του τελεστή αναφέρονται στον προορισμό. Υπάρχει επίσης και ο αμφίδρομος τελεστής <>, ο οποίος λέει στο Snort ότι δεν ενδιαφέρει η κατεύθυνση της κίνησης, όπως για παράδειγμα:

```
alert tcp ![192.168.1.0/24,10.1.1.0/24] any <> 192.168.1.0/24 23
```

3.3.6 Ενεργοί και δυναμικοί κανόνες

Τα ζευγάρια των ενεργών/δυναμικών κανόνων δίνουν στο Snort την εξής πολύ σημαντική ιδιότητα: επιτρέπουν σε ένα κανόνα να ενεργοποιεί έναν άλλο όταν η ενέργεια του εκτελείται για ένα αριθμό πακέτων. Αυτό είναι πολύ σημαντικό, όταν κάποιος θέλει να παραμετροποιήσει το Snort, να εκτελέσει μια καταγραφή όταν ένας κανόνας ενεργοποιηθεί. Οι ενεργοί κανόνες λειτουργούν σαν κανόνες προειδοποιήσης, εκτός του, το οτι έχουν ένα υποχρεωτικό πεδίο *activates*. Οι δυναμικοί κανόνες λειτουγούν σαν κανόνες καταγραφής, μόνο που έχουν ένα υποχρεωτικό πεδίο *activated_by* και ένα επίσης υποχρεωτικό πεδίο *count*. Για παράδειγμα οι δύο επόμενοι κανόνες λένε στο Snort, όταν ανιχνεύσει μια υπερχείλιση καταχωρητή IMAP, να συλλέξει καταγράψει τα επόμενα 50 πακέτα που προορίζονται για την πόρτα 143 του δικτύου και προέρχονται από το εξωτερικό του δικτύου.

```
activate tcp !$HOME_NET any -> $HOME_NET 143 (flags: PA;
  content:"|E8C0FFFFFF|/bin"; activates:1;
  msg: "IMAP buffer overflow";)
dynamic tcp !$HOME_NET any -> $HOME_NET 143 \
    ( activated_by:1; count:50;)
```

3.4 Οι ρυθμίσεις των κανόνων

Οι ρυθμίσεις των κανόνων αποτελούν την καρδιά της μηχανής ανίχνευσης του Snort, συνδυάζοντας ευκολία χρήσης με ευελιξία και ισχύ. Όλες οι ρυθμίσεις διαχωρίζονται μεταξύ τους με το χαρακτήρα ; . Οι λέξεις κλειδιά διαχωρίζονται από τα ορίσματα τους με το χαρακτήρα : . Υπάρχουν τέσσερις κατηγορίες ρυθμίσεων:

Βοηθητικές παρέχουν πληροφορίες για τον κανόνα αλλά δεν έχουν καμία επίδραση κατά την εκτέλεση του κανόνα

Περιεχομένου ψάχνουν για δεδομένα μέσα στο σώμα του πακέτου και μπορεί να συσχετίζονται μεταξύ τους

Μη-Περιεχομένου ψάχνουν για δεδομένα εκτός του σώματος του πακέτου

Ολοκληρώμενης αναζήτησης προσδιορίζουν κάποια γεγονότα μετά την επαλήθευση κάποιου κανόνα.

3.4.1 Βοηθητικές ρυθμίσεις

- msg - καθορίζει το μήνυμα που θα εμφανίσουν τα plugins εξόδου
- reference - επιτρέπει στους κανόνες να περιλαμβάνουν αναφορές σε εξωτερικά συστήματα NIDS(όπως τα bugtrag,cve και άλλα) καθώς επίσης και μοναδικά URLs.
- sid - είναι το αναγνωριστικό του κανόνα και χρησιμοποιείται συνήθως μαζί με το rev(βλέπε επόμενο bullet) για να αναγνωρίζει μοναδικά τον κάθε κανόνα.
- rev - χρησιμοποιείται για να αναγνωρίζει μοναδικά εκδόσεις του ίδιου κανόνα. Σε συνδυασμό με τη sid επιτρέπουν την ανανέωση των κανόνων με νέες πληροφορίες

- classtype - χρησιμοποιείται για να κατηγοριοποιήσει ένα κανόνα που ανιχνεύει μια επίθεση, σε μια κλάση γνωστών επιθέσεων. Το Snort παρέχει ένα σύνολο κλάσεων αλλά οι χρήστες μπορούν να ορίσουν τις δικές τους κλάσεις ανάλογα με τη εφαρμογή. Το default σύνολο κλάσεων του Snort ορίζει και μια προτεραιότητα στις κλάσεις η οποία μπορεί να επναοριστεί από το χρήστη(βλέπε επόμενο bullet)

- priority - ορίζει τη προτεραιότητα ενός κανόνα. Χρησιμοποιείται, επίσης, σε συνδυασμό με τη classtype και επιτρέπει τον επαναορισμό της προτεραιότητας μιας κλάσης.

- metadata - επιτρέπουν στον συγγραφέα των κανόνων να ενσωματώσει επιπλέον πληροφορίες σχετικά με τον κανόνα, με τη μορφη *κλειδί-τιμή*.

3.4.2 Περιεχομένου

- content - από τις πιο σημαντικές ρυθμίσεις των κανόνων του Snort. Επιτρέπει στο χρήστη να ορίσει κανόνες που ψάχνουν για συγκεκριμένες συμβολοσειρές στο σώμα του πακέτου. Η αναζήτηση της συμβολοσειράς γίνεται με τον αλγόριθμο Boyer-Moore. Αν η αναζήτηση είναι επιτυχής τότε ελέχγονται οι υπόλοιπες ρυθμίσεις του κανόνα. Να σημειωθεί ότι ο default τρόπος αναζήτησης της συμβολοσειράς είναι case sensitive. Επιπλέον υπάρχουν ορισμένες λέξεις κλειδιά που τροποποιούν την default λειτουργία της content, όπως φαίνεται παρακάτω:

  - nocase - επιτρέπει non case sensitive αναζήτηση
  - rawbytes - επιτρέπει στους κανόνες να κοιτάνε δεδομένα πακέτου χωρίς να έχουν υποστεί καμία αποκωδικοποίηση
  - depth - καθορίζει πόσο βαθιά στο πακέτο θα ψάξει το Snort για μια συγκεκριμένη συμβολοσειρά. Για παράδειγμα, αν depth:5, τότε θα εξεταστούν τα 5 πρώτα bytes του σώματος του πακέτου.
  - offset - καθορίζει το σημείο από το οποίο θα ξεκινήσει η αναζήτηση. Για παράδειγμα, αν offset:5, η αναζήτηση θα ξεκινήσει από το πέμπτο byte του σώματος του πακέτου.
  - distance - είναι ακριβώς ίδιο με το depth εκτός του ότι είναι σχετικό με το τέλος της προηγούμενης αναζήτησης
  - within - αναφέρεται στο πλήθος των bytes μεταξύ δύο contents. Χρησιμοποιείται συνήθως σε συνδυασμό με την distance
  - http_client_body - χρησιμοποιείται για να περιορίσει την αναζήτηση στο κανονικοποιημένο σώμα μια αίτησης ενός πελάτη HTTP.

  - http_uri - χρησιμοποιείται για να περιορίσει την αναζήτηση σε μια κανονικοποιημένη αίτηση URI

- uricontent - η ρύθμιση αυτή ψάχνει το κανονικοποιημένο πεδίο μια αίτησης URI. Αυτό σημαίνει ότι αν κάποιος γράφει κανόνες που περιέχουν για παράδειγμα %2f ή διαδρομές καταλόγου, αυτοί οι κανόνες δεν θα ενεργοποιηθούν.

- isdataat - επικυρώνει ότι το σώμα του πακέτου έχει δεδομένα σε μια συγκεκριμένη θέση. Προαιρετικά, αυτή η ρύθμιση ψάχνει για δεδομένα σε σχέση με το τέλος της προηγούμενης αναζήτησης.

- pcre - επιτρέπει να γράφονται κανόνες χρησιμοποιώντας κανονικές εκφράσεις συμβατές με τη γλώσσα perl.

- byte_test - συγκρίνει ένα πεδίο από bytes με μια συγκεκριμένη τιμή, χρησιμοποιώντας τελεστές όπως <,>,!,&

- byte_jump - η εντολή αυτή μας επιτρέπει να εξάγουμε ορισμένα bytes από τα δεδομένα, να τα μετατρέπουμε στον αντίστοιχο αριθμό και να εκτελούμε ένα άλμα κατά τόσα bytes για περαιτέρω περιεχομένου ή σύγκριση bytes. Με τον τρόπο αυτό, μπορούν να δημιουργηθούν μέθοδοι ανίχνευσης που λαμβάνουν υπόψη τις αριθμιτικές τιμές που περιέχονται στα δεδομένα.

3.4.3 Ρυθμίσεις μη-περιεχομένου

- fragoffset - επιτρέπει τη σύγκριση του πεδίου fragment offset της κεφαλίδας IP με αριθμιτική τιμή.

- ttl - χρησιμοποιείται για να ελέγχει την τιμή του πεδίου time-to-live σε μια κεφαλίδα IP

- tos - χρησιμοποιείται για να ελέγχει την τιμή του πεδίου TOS σε μια κεφαλίδα IP

- id - χρησιμοποιείται για να ελέγχει την τιμή του πεδίου ID σε μια κεφαλίδα IP

- ipopts - χρησιμοποιείται για να ελέγχει αν μια συγκεκριμένη επιλογή IP είναι παρούσα. Οι έγκυρες τιμές της ρύθμισης αυτής είναι:

  - rr - Record Route
  - eol - End of list

  - nop - No Op
  - ts - Time Stamp
  - sec - IP Security
  - esec - IP Extended Security
  - lsrr - Loose Source Routing
  - ssrr - Strict Source Routing
  - satid - Stream identifier
  - any - οποιαδήποτε επιλογή IP

- fragbits - χρησιμοποιείται για να ελέγχει αν έχουν οριστεί τα bit κατακερματισμού στη κεφαλίδα IP

- dsize - χρησιμοποιείται για να ελέγχει το μέγεθος του σώματος των πακέτων. Μπορεί να χρησιμοποιηθεί για να ανιχνέυσει πακέτα ακανόνιστου μεγέθους και για υπερχείλιση ενταμιευτή

- flags - χρησιμοποιείται για να ελέγχει αν συγκεκριμένες σημαίες TCP είναι παρούσες. Τα ακόλουθα bit μπορούν να ελεγχθούν:

  - F - FIN
  - S - SYN
  - R - RST
  - P - PSH
  - A - ACK
  - U - URG
  - 1 - Δεσμευμένο bit 1
  - 2 - Δεσμευμένο bit 2
  - 0 - Δεν υπάρχουν σημαίες TCP παρούσες

- flow - χρησιμοποιείται σε συνδυασμό με τον προεπεξεργαστή επανασύνθεσης ροών TCP(TCP stream reassembly). Επιτρέπει την εφαρμογή του κανόνα μόνο σε συγκεκριμένη κατεύθυνση της ροής, για παράδειγμα στη ροή προς τον πελάτη

- flowbits - επιτρέπει στους κανόνες να ανιχνεύουν καταστάσεις ανάμεσα στις συνόδους του πρωτοκόλλου μεταφοράς. Η ρύθμιση αυτή είναι κυρίως χρήσιμη στις συνόδους TCP, καθώς επιτρέπει στους κανόνες να ανιχνεύουν τη κατάσταση του πρωτοκόλλου εφαρμογής

- seq - χρησιμοποιείται για να ελέγχει τον αριθμό ακολουθίας ενός TCP πακέτου
- ack - χρησιμοποιείται για να ελέγχει τον αριθμό επιβεβαίωσης ενός TCP πακέτου
- window - χρησιμοποιείται για να ελέγχει το μέγεθος παραθύρου ενός TCP πακέτου
- itype - χρησιμοποιείται για να ελέγχει μια συγκεκριμένη τιμή ICMP
- icode - χρησιμοποιείται για να ελέγχει ένα συγκεκριμένο κωδικό ICMP
- icmp_id - χρησιμοποιείται για να ελέγχει μια συγκεκριμένη τιμή ICMP ID
- icmp_seg - χρησιμοποιείται για να ελέγχει ένα συγκεκριμένο αριθμό ακολουθίας ICMP
- rpc - χρησιμοποιείται για να ελέγχει μια εφαρμογή RPC,έκδοση και αριθμούς διαδικασίας σε SUNRPC CALL αιτήσεις
- ip_proto - επιτρέπει τη σύγκριση του αριθμού του IP ορισμένες τιμές
- sameip - εξετάζει αν η διεύθυνση αποστολέα και παραλήπτη είναι ίδιες

3.4.4 Ρυθμίσεις Ολοκληρωμένης Αναζήτησης

- logto - χρησιμοποιείται για την καταγραφή όλων των πακέτων που ενεργοποιούν ένα κανόνα σε ένα ειδικό αρχείο καταγραφής
- session - χρησιμοποιείται για να εξάγει δεδομένα χρηστών από συνόδους TCP
- resp - χρησιμοποιείται για να κλείσει όλες τις συνόδους όταν ενεργοποιείται μια προειδοποίηση(alert). Στην ορολογία του Snort αυτό λέγεται *ευέλικτη απάντηση*
- react - επιτρέπει στους χρήστες να αντιδρούν σε κίνηση που ενεργοποιεί ένα συγκεκριμένο κανόνα. Η πιο συνηθισμένη σντίδραση είναι το μπλοκάρισμα των ιστοσελίδων που θέλουν να προσπελάσουν οι χρήστες. Η ρύθμιση αυτή επιτρέπει στο Snort να κλείνει επιθετικές/επικίνδυνες συνδέσεις ή να στέλνει προειδοποιητικά μηνύματα στον περιηγητή.

- tag - επιτρέπει στο χρήστη να καταγράφει περισσότερα από το πακέτο που ενεργοποιήσε ένα κανόνα. Από τη στιγμή που ένας κανόνας ενεργοποιείται, επιπλέον κίνηση που αφορά τη προέλευση ή/και τον προορισμό καταγράφεται

3.5 Η μηχανή ανίχνευσης του Snort

Η μηχανή ανίχνευσης αποτελεί την καρδιά του μηχανισμού του Snort. Σκοπός της είναι, να ανιχνεύσει ύποπτες συμβολοσειρές στο σώμα των πακέτων, λαμβάνοντας υπόψη τους κανόνες του Snort. Υπό αυτή την έννοια, σκοπός της μηχανής ανίχνευσης είναι να εφαρμόσει τους κανόνες στη διερχόμενη κίνηση. Η μηχανή ανίχνευσης αποτελείται από τρία κύρια κομμάτια, τον Βελτίστοποιητή Κανόνων (Rule Optimizer) και την Μηχανή Επιθεώρησης Πολλαπλών κανόνων (High Perfomance Multi-Rule Inspection Engine) και τον Επιλογέα Συμβάντων(Event Selector).

3.5.1 Ο Βελτιστοποιητής Κανόνων

Ο Βελτιστοποιητής Κανόνων αποτελεί σημαντικό κομμάτι της μηχανής ανίχνευσης του Snort από την έκδοση 2.0 και μετά. Σκοπός του είναι, να επιταχύνει τη λειτουργία του Snort και αυτό το καταφέρνει, οργανώνοντας τους κανόνες σε ομάδες με τέτοιο τρόπο ώστε κάθε πακέτο να συγκρίνεται με μόνο μια ομάδα και η σύγκριση αυτή να μπορεί να γίνεται με τρόπο παράλληλο.

Η επεξεργασία των Κανόνων πριν την έκδοση 2.0 του Snort

Στις προηγούμενες εκδόσεις του Snort, η επεξεργασία των κανόνων γινόταν, με μια παραμετρική αναζήτηση όλων των κανόνων όπου οι παράμετροι της αναζήτησης ήταν οι ρυθμίσεις των κανόνων. Μία καλή παραμετρική αναζήτηση θα χρησιμοποιούσε μια σειρά αναζήτησης, η οποία θα τερμάτιζε μια ανεπιτυχή αναζήτηση όσο το δυνατό γρηγορότερα, ή θα περιόριζε αποδοτικά των χώρο αναζήτησης της επόμενη παραμέτρου. Γενικότερα σε ένα πρόβλημα παραμετροποιήσης, η σειρά των παραμέτρων καθορίζεται από την πληροφορία που παρέχει μια παράμετρος στο να τερματίζει γρήγορα μια ανεπιτυχή αναζήτηση ή να διασφαλίζει μια επιτυχή αναζήτηση, και βασίζεται σε βαθύτερη κατανόηση του προβλήματος το οποίο παραμετροποιείται. Συγκεκριμένα το Snort, βασίζεται στα χαρακτηριστικά του πρωτοκόλλου IP για να καθορίσει αποδοτικά τη σειρά των παραμέτρων.

Ιστορικά, το Snort χρησιμοποιούσε μια αμιγώς παραμετρική αναζήτηση για να ομαδοποιήσει τους κανόνες και να αποφασίσει αν ένα πακέτο ή μια ροή

πακέτων ταιριάζει με τους παραμέτρους μιας ομάδας. Το Snort ομαδοποιούσε τους κανόνες σε ομάδες χρησιμοποιώντας τέσσερις παραμέτρους:

- Διεύθυνση IP αποστολέα
- Διεύθυνση IP παραλήπτη
- Πιθανές τιμές πόρτας αποστολέα
- Πιθανές τιμές πόρτας παραλήπτη

Όταν ένα πακέτο εξεταζόταν, αρχικά συγκρίνοταν με τις τέσσερις παραμέτρους μιας ομάδας για να αποφασιστεί με ποιά ομάδα θα εξεταστεί. Στην συνέχεια όταν βρισκόταν μια ομάδα, οι κανόνες της ομάδας εξετάζονταν ακολουθιακά και εναπομείναντες παράμετροι κάθε κανόνα στη σειρά. Αφού εξεταζόντουσαν όλοι οι κανόνες της ομάδας η διαδικασία επαναλαμβανόταν με την επόμενη ομάδα.

Είναι προφανές, ότι η συγκεκριμένη διαδικασία δεν είναι αποδοτική για μεγάλο πλήθος κανόνων, ειδικά αν αναλογιστεί κανείς ότι συχνά απαιτείται η εξέταση του κάθε πακέτου με πολλές διαφορετικές ομάδες.

Για να λυθεί το παραπάνω πρόβλημα, έπρεπε να υλοποιηθούν ομαδικές μέθοδοι επεξεργασίας (set-based methodologies). Επίσης για να είναι οι μέθοδοι αυτοί αποδοτικές πρέπει η εξέταση κάθε πακέτου να γίνεται με μόνο μια ομάδα κάθε φορα. Αυτό είναι η αποκλειστική δουλειά του Βελτιστοποιητή Κανόνων, η δημιουργία αυτών των ομάδων.

Η επεξεργασία κανόνων μετά την έκδοση 2.0 του Snort

Ο βελτιστοποιητής κανόνων δημιουργεί ομάδες κανόνων, οι οποίες είναι κατάλληλες για ομαδικές μεθοδολογίες επιθεώρησης. Για να επιτευχθεί αυτός ο στόχος, ο βελτιστοποιητής κανόνων πρέπει να ικανοποιεί τις εξής απαιτήσεις:

1. Να δημιουργεί τις μικρότερες, περισσότερο αποδοτικές ομάδες
2. Να δημιουργεί διακεκριμένες ομάδες, με τέτοιο τρόπο, ώστε για κάθε εισερχόμενο πακέτο να εξετάζεται μόνο με μια ομάδα

Ο βελτιστοποιητής κανόνων δημιουργεί αυτές τις ομάδες κατά τη διάρκεια της αρχικοποίησης χρησιμοποιώντας τις πιο μοναδικές παραμέτρους των κανόνων του Snort. Οι επιλεγμένες παράμετροι είναι διαφορετικές για κάθε πρωτόκολλο μεταφοράς, επειδή κάθε πρωτόκολλο μεταφοράς έχει διαφορετικές παραμέτρους που το κάνουν μοναδικό. Για παράδειγμα, ένας κανόνας TCP μπορεί να είναι μοναδικός από έναν άλλο κανόνα TCP βασιζόμενος στην πόρτα αποστολής και στην πόρτα προορισμού, ενώ ένας ICPM κανόνας μπορεί να

μοναδικός βασιζόμενος στον τύπο ICPM αυτού του κανόνα. Ο βελτιστοποιητής κανόνων παίρνει όλες αυτές τις μοναδικές παραμέτρους και σχηματίζει υποσύνολα κανόνων που βασίζονται σε αυτά. Αυτό δίνει στην μηχανή ανίχνευσης πολύ μικρότερους κανόνες να επιθεωρήσει. Και πιο σημαντικό είναι ότι επιτρέπει το κάθε πακέτο να ταξινομείται σε ένα υποσύνολο κανόνων χρησιμοποιώντας τα χαρακτηριστικά του πακέτου.

Κατά τη διάρκεια της εκτέλεσης του Snort, ο βελτιστοποιητής κανόνων επιλέγει μια ομάδα για κάθε πακέτο που επεξεργάζεται. Για κάποια ανώμαλα πακέτα, υπάρχει η επιλογή να επιλεχθούν 2 ομάδες. Αυτή η περίπτωση λέγεται μοναδική σύγκρουση και εξετάζεται περισσότερο παρακάτω. Είναι σημαντικό να σημειωθεί ότι ο βελτιστοποιητής κανόνων είναι το πρώτο στάδιο της μηχανής επιθεώρησης πολλαπλών κανόνων. Με αυτή την έννοια, ο βελτιστοποιητής κανόνων ενεργοποιεί μόνο τους κανόνες που θα μπορούσαν να ταιριάζουν με αυτό το πακέτο, και φιλτράρει μόνο τους κανόνες που χρειάζεται να περάσουν από τη διαδικασία της μηχανής επιθεώρησης πολλαπλών κανόνων.

Ο βελτιστοποιητής κανόνων στην πράξη. Ο βελτιστοποιητής κανόνων διαιρεί τους κανόνες σε διακριτά, σύνολα τα οποία το Snort μπορεί να προσπελάσει γρήγορα. Αυτό επιτυγχάνεται χρησιμοποιώντας τις ακόλουθες μοναδικές παραμέτρους που εξαρτώνται από το πρωτόκολλο μεταφοράς, όπως φαίνεται στον παρακάτω πίνακα 3.1

Πίνακας 3.1: Παράμετροι που χρησιμοποιούνται στη κατηγοριοποίηση των κανόνων στο Snort

TCP/UDP	Πόρτα αποστολής Πόρτα προορισμού
ICMP	Τύπος ICMP
IP	Πρωτόκολλο μεταφοράς id

TCP/UDP Οι πιο μοναδικές ιδιότητες των TCP/UDP πρωτόκολλων είναι οι πόρτες προορισμού και λήψης. Οι πόρτες διαιρούνται σε 2 γενικές κατηγορίες: δεσμευμένες και μη δεσμευμένες. Αυτό σημαίνει ότι τα περισσότερα είδη επικοινωνίας μεταξύ των hosts θα έχουν μια δεσμευμένη πόρτα (συνήθως τον server) με την άλλη πόρτα μη δεσμευμένη (πάνω από 1024 και συνήθως τον πελάτη). Αυτή η ιδιότητα επιτρέπει στον βελτιστοποιητή κανόνων να χρησιμοποιεί την δεσμευμένη πόρτα ως μοναδική παράμετρο. Χρησιμοποιώντας την δεσμευμένη πόρτα το Snort επιλέγει ομάδες που βασίζονται στο αν η δεσμευμένη πόρτα βρίσκεται είτε στην πόρτα αποστολής είτε στην πόρτα λήψης. Αν

η δεσμευμένη πόρτα βρίσκεται στην πόρτα προορισμού, αυτό συνήθως σημαίνει ότι η κίνηση είναι από τον πελάτη προς τον εξυπηρετητή. Παρατηρώντας τα πρωτόκολλα TCP/UDP στην πράξη βοηθάει να επιδειχθεί αυτή η άποψη. Το TCP/UDP είναι μια επικοινωνία 2 δρόμων, μεταξύ του πελάτη και του εξυπηρετητή. Αυτό σημαίνει ότι το Snort βλέπει τα πακέτα από τον πελάτη που πάνε στον εξυπηρετητή και από τον εξυπηρετητή στον πελάτη. Έτσι αντί να εφαρμόζει όλους τους κανόνες για την επικοινωνία και του πελάτη και του εξυπηρετητή σε ένα πακέτο, ο βελτιστοποιητής πακέτων ομαδοποιεί τους κανόνες που βασίζονται στην τοποθεσία της μοναδικής πόρτας, είτε στην πόρτα αποστολής είτε στην πόρτα λήψης. Μ'Α αυτόν τον τρόπο, οι κανόνες που αναφέρονται σε αίτηση του πελάτη και οι κανόνες που αναφέρονται σε απόκριση του server είναι δύο μοναδικές ομάδες. Με αυτό το τρόπο τα πακέτα πχ, του πελάτη, εξετάζονται μόνο από τους κανόνες αίτησης του πελάτη και αντίστοιχα για τον server. Για παράδειγμα η κίνηση μιας HTTP αίτησης του πελάτη πρέπει να επιθεωρείται ενάντια στους HTTP αίτηση-πελάτη κανόνες, όχι στους HTTP απόκριση-server κανόνες. Έτσι, όταν ένα πακέτο προέρχεται από ένα HTTP πελάτη (το οποίο σημαίνει ότι η πόρτα 80 βρίσκεται στο πεδίο της TCP πόρτας προορισμού), μαζί οι πόρτες αποστολής και λήψεις ελέγχονται αν είναι δεσμευμένες. Σχεδόν πάντα για την κίνηση HTTP του πελάτη, η πόρτα αποστολής δεν είναι δεσμευμένη πόρτα (γιατί είναι μεγαλύτερη από 1024), αλλά η πόρτα προορισμού είναι δεσμευμένη μόνο αν προορίζεται για έναν HTTP server, ο οποίος διαμένει σε μια πόρτα 80. Έτσι, σε περίπτωση ενός HTTP πελάτη πακέτου, η ομάδα με τις παραμέτρους της πόρτας προορισμού 80 και η γενική πόρτα αποστολής επιλέγεται και επιθεωρείται από τη μηχανή επιθεώρησης πολλαπλών κανόνων.

ICMP Οι ICMP κανόνες χρησιμοποιούν μόνο το πεδίο του ICMP τύπου για τη βελτιστοποίηση των ICMP ομάδων. Οι κανόνες που δεν έχουν ένα πεδίο τύπου θεωρούνται γενικοί ICMP κανόνες και προστίθενται σε κάθε ομάδα, συμπεριλαμβάνοντας τη γενική ομάδα. Για παράδειγμα, ένα εισερχόμενο ICMP πακέτο με κωδικό τύπου 8 (echo request) θα επιθεωρηθεί ενάντια στον κωδικό τύπου 8 σαν παράμετρο. Παρόλα αυτά αν ο κωδικός τύπου δεν είναι μοναδικός, επιθεωρείται ενάντια στη γενική ICMP ομάδα.

IP Ο βελτιστοποιητής κανόνων υποστηρίζει το πρωτόκολλο IP χρησιμοποιώντας το πεδίο του πρωτοκόλλου μεταφοράς στην κεφαλίδα IP . Όλοι οι IP κανόνες που περιλαμβάνουν TCP, UDP ή ICMP ομαδοποιούνται στην αντίστοιχη ομάδα πρωτοκόλλου. Όμως τα υπόλοιπα πρωτόκολλα που δεν υποστηρίζονται ομαδοποιούνται σε IP ομάδες που βασίζονται στο πεδίο μεταφοράς του πρωτοκόλλου. Κανόνες που δεν περιέχουν κανένα πεδίο πρωτοκόλλου θεωρούνται

γενικοί IP κανόνες και προστίθενται σε κάθε IP, TCP, UDP και ICMP ομάδα.

Στην πράξη ο βελτιστοποιητής κανόνων χρησιμοποιεί την πληροφορία του πρωτοκόλλου για να δημιουργήσει και επιλέξει αποδοτικές ομάδες. Αλλά ο βελτιστοποιητής πρέπει επίσης να επιλέξει μόνο μια ομάδα ανά πακέτο. Για να επιλέξει μόνο μια ομάδα, πρέπει να συνδυάσει τους μοναδικούς κανόνες για αυτήν την ομάδα με τους γενικούς κανόνες για το πρωτόκολλο μεταφοράς των μοναδικών κανόνων.

Μοναδικοί κανόνες ενάντια στους γενικούς κανόνες Οι μοναδικοί κανόνες καθορίζονται από την ύπαρξη μοναδικών παραμέτρων. Αυτές οι μοναδικές παράμετροι μπορούν να είναι συγκεκριμένες πληροφορίες στις κεφαλίδες μεταφοράς ή στα IPs στα οποία αναφέρεται ο κανόνας. Οι γενικοί κανόνες δεν έχουν καμία μοναδική παράμετρο που να τους ξεχωρίζει από άλλους κανόνες. Για παράδειμα, ένας κανόνας μπορεί να γραφτεί και να λέει *Ταίριαξε αυτό το περιεχόμενο σε όλα τα πακέτα.* Σε αυτή την περίπτωση, οι παράμετροι των κανόνων δεν είναι μοναδικές, επειδή δεν υπάρχει τίποτα εκτός του περιεχομένου που να ξεχωρίζει αυτόν τον κανόνα από οποιονδήποτε άλλο κανόνα.

Και οι μοναδικοί και οι γενικοί κανόνες πρέπει να έχουν ξεχωριστές ομάδες. Χρησιμοποιώντας τις μοναδικές ομάδες και τα καλύμματα των γενικών ομάδων καλύπτονται και τα δύο σενάρια επιλογής των ομάδων. Δημιουργώντας μια μοναδική ομάδα καλύπτεται το σενάριο, όπου ένα πακέτο ταιριάζει με μια μοναδική ομάδα. Αντίστοιχα, δημιουργώντας μια γενική ομάδα καλύπτεται το σενάριο, όπου ένα πακέτο δεν ταιριάζει με μια μοναδική ομάδα επειδή το πακέτο πρέπει ακόμα να επιθεωρηθεί ενάντια σε όλους τους γενικούς κανόνες για αυτό το πρωτόκολλο.

Έτσι για να λειτουργήσει αυτός ο διαχωρισμός των μοναδικών κανόνων και των γενικών κανόνων, οι γενικοί κανόνες πρέπει να προστεθούν σε όλες τις μοναδικές ομάδες. Διαφορετικά, δύο ομάδες θα έπρεπε να αναζητηθούν για κάθε πακέτο, η μοναδική και η γενική ομάδα. Από τη στιγμή που ένα πακέτο ταιριάζει σε μια μοναδική ομάδα και μπορεί επίσης να ταιριάζει επίσης σε ένα γενικό κανόνα, αυτές οι δύο ομάδες συνδυάζονται.Έτσι, ο βελτιστοποιητής κανόνων επιλέγει είτε μια μοναδική είτε μια γενική ομάδα, ανάλογα στο πακέτο που επιθεωρείται. Τι γίνεται όμως αν ένα πακέτο ταιριάζει σε πολλούς μοναδικούς κανόνες? Αυτή η συνθήκη λέγεται μοναδική σύγκρουση.

Μοναδικές συγκρούσεις Μοναδικές συγκρούσεις προκαλούνται όταν υπάρχουν 2 ή περισσότερες παράμετροι από τις οποίες επιλέγονται οι ομάδες. Αν και οι δύο από αυτές τις παραμέτρους είναι μοναδικές, τότε υπάρχει μοναδική σύγκρουση. Προσωρινά, αυτό συμβαίνει μόνο σε TCP και UDP κανόνες. Υπάρχουν δύο τύποι μοναδικών συγκρούσεων: καθορισμένες μοναδικές

συγκρούσεις και μη καθορισμένες μοναδικές συγκρούσεις. Οι καθορισμένες μοναδικές συγκρούσεις χειρίζονται μέσω επιθεώρησης και των δύο νόμιμων μοναδικών ομάδων συνδυασμένες σε μια ομάδα. Οι μη καθορισμένες μοναδικές συγκρούσεις χειρίζονται από έναν αριθμό από μεθόδους και η μέθοδος που χρησιμοποιείται εξαρτάται από το επίπεδο της εκτέλεσης που απαιτεί ο χρήστης και εξετάζονται παρακάτω. Δεν μπορούν όλες οι μοναδικές συγκρούσεις να μετατραπούν σε καθορισμένες επειδή χώρος διευθύνσεων είναι πολύ μεγάλος. Για παράδειγμα, αν υπάρχουν 1000 ομάδες με μοναδική πόρτα αποστολής και 1000 ομάδες με μοναδική πόρτα προορισμού, τότε θα έπρεπε να υπάρχουν 1.000.000 ομάδες με μοναδικές πόρτες σύγκρουσης. Σκέφτοντας τις απαιτήσεις της μνήμης για μια τέτοια ομάδα, αυτό δεν είναι πρακτικό. Ενώ οι καθορισμένες μοναδικές συγκρούσεις συμβαίνουν φυσικά, οι μη καθορισμένες συμβαίνουν σπάνια και όταν αυτό γίνεται προκαλεί το ενδιαφέρον.

Καθορισμένες μοναδικές συγκρούσεις Ένα παράδειγμα καθορισμένης μοναδικής σύγκρουσης παρουσιάζεται στα ερωτήματα DNS μεταξύ των εξυπηρετητών ονομάτων(DNS servers). Κατά τη διάρκεια της και οι δύο πόρτες θα είναι 53, έτσι εμφανίζεται το πρόβλημα της επιλογής της ομάδας που θα εφαρμοστεί σε αυτό το πακέτο. Η απάντηση είναι και οι δύο. Οι καθορισμένες μοναδικές συγκρούσεις καθορίζονται από τον προορισμό του πρωτόκολλου και σε αυτές τις περιπτώσεις υπάρχει μια ξεχωριστή ομάδα από κανόνες που περιέχουν και κανόνες του πελάτη και του εξυπηρετητή. Αυτό σημαίνει ότι όλοι οι κανόνες που μπορούν να εφαρμοστούν σε μια μοναδική σύγκρουση εφαρμόζονται σε ένα πέρασμα αντί για δύο ξεχωριστά.

Μη Καθορισμένες μοναδικές συγκρούσεις Ενώ οι καθορισμένες μοναδικές συγκρούσεις συμβαίνουν φυσικά, οι μη καθορισμένες συμβαίνουν σπάνια. Αν συμβαίνουν μη καθορισμένες, αυτό γίνεται συνήθως γιατί υπάρχει κάποιο κακοήθες ή ανώμαλο συμβάν. Παρόλα αυτά, όταν συμβαίνουν μη καθορισμένες μοναδικές συγκρούσεις, μια εναλλακτική ροή επιλογής ομάδας πρέπει να λάβει χώρα. Ο λόγος είναι γιατί μια μη καθορισμένη μοναδική σύγκρουση έχει ως αποτέλεσμα 2 ομάδες που πρέπει αν επιθεωρηθούν. Υπάρχουν μερικές διαφορετικές μέθοδοι χειρισμού μοναδικών συγκρούσεων: διπλή επιθεώρηση, first-hit επιθεώρηση και πιθανολογική επιθεώρηση.

1. Διπλή επιθεώρηση - Η διπλή επιθεώρηση είναι ο πιο βασικός τύπος μεθόδου χειρισμού μιας μη καθόρισμένης μοναδικής σύγκρουσης. Από τη στιγμή που υπάρχουν 2 ομάδες για μια μη καθορισμένη μοναδική σύγκρουση, η μέθοδος της διπλής επιθεώρησης απλά επιθεωρεί και τις 2 ομάδες, διπλασιάζοντας κατά προσέγγιση τον χρόνο που κάνει μια επιθεώρηση. Αυτή η μέθοδος εγγυάται ότι όλα τα συμβάντα θα βρεθούν

σε ένα πακέτο, παρόλα αυτά η μέθοδος αυτή είναι πιο επιρρεπής σε επιθέσεις DOS(Denial of service) , εξαναγκάζοντας την διπλή επιθεώρηση με κατασκευασμένα πακέτα που προκαλούν μη καθορισμένες μοναδικές συγκρούσεις.

2. First-hit επιθεώρηση - Η μέθοδος της first-hit επιθεώρησης είναι όμοια με της διπλής επιθεώρησης καθώς επιθεωρεί και τις 2 ομάδες. Η διαφορά είναι ότι αν επαληθευτεί η ομάδα που επιθεωρήθηκε πρώτα, τότε η δεύτερη ομάδα δεν επιθεωρείται ποτέ. Αυτό βοηθάει την επίδοση από την μέθοδο διπλής επιθεώρησης

3. Πιθανολογική επιθεώρηση - Η μέθοδος της πιθανολογικής επιθεώρησης επιλέγει μια από τις 2 ομάδες τυχαία. Χρησιμοποιώντας αυτή τη μέθοδο, εμποδίζονται οι πιθανές επιθέσεις DOS χάρη στις μη καθορισμένες μοναδικές συγκρούσεις. Αλλά δεν εγγυάται ότι αν υπάρχουν συμβάντα στο πακέτο, αυτά θα βρεθούν. Παρόλα αυτά, επίσης προστατεύει ενάντια σε συτόν που επιτίθεται χρησιμοποιώντας τη γνώση ότι μόνο μια ομάδα θα επιλεχθεί σε μια μη καθορισμένη μοναδική σύγκρουση. Για παράδειγμα, αυτός που επιτίθεται θα μπορούσε να εκμεταλλευτεί μια μέθοδο που επιλέγει είτε τη μια είτε την άλλη ομάδα για μη καθορισμένες μοναδικές συγκρούσεις. Αυτός που επιτίθεται θα μπορούσε να χρησιμοπιήσει μια επίθεση που υπάρχει σε μια ομάδα που δεν πρόκειται να επιθεωρηθεί κατά την πρόκληση μη καθορισμένης μοναδικής σύγκρουσης. Χρησιμοποιώντας αυτή τη μέθοδο, αυτός που επιτίθεται χρειάζεται να ρίξει τα ζάρια και δεν μπορεί αν είναι ποτέ σίγουρος ότι η επίθεσή τους ανιχνεύθηκε ή όχι

Από τις 3 αυτές μεθόδους, η first-hit επιθεώρηση είναι η μέση λύση. Βοηθάει στην αντιμετώπιση των επιθέσεων DOS χάρη στη μέθοδο διπλής επιθεώρησης και επίσης, δημιουργεί ένα συμβάν (αν είναι δυνατόν) για κάθε μη καθορισμένη μοναδική σύγκρουση. Παρόλα αυτά, δεν εγγυάται την αντιμετώπιση επιθέσεων DOS σε σχέση με την πιθανολογική επιθεώρηση, ούτε και εγγυάται ότι κάθε συμβάν θα βρεθεί όπως με την μέθοδο της διπλής επιθεώρησης.

3.5.2 Η μηχανή επιθεώρησης πολλαπλών κανόνων

Η μηχανή επιθεώρησης πολλαπλών κανόνων μπορεί να εκτελέσει τρεις διαφορετικές κατηγορίες αναζητήσεων ανάλογα με τις ιδιότητες των μοναδικών κανόνων του Snort.

Αναζήτηση πεδίου πρωτοκόλλου Επιτρέπει σε ένα κανόνα να προσδιορίζει ένα συγκεκριμένο πεδίο ενός πρωτοκόλλου μέσα στο οποίο ψάχνουμε

για ύποπτο περιεχόμενο. Για παράδειγμα, το Snort χρησιμοποιεί τη λέξη κλειδί uricontent, για την αναζήτηση μέσα στα πεδία URI μιας αίτησης HTTP

Αναζήτηση γενικού περιεχομένου Επιτρέπει σε ένα κανόνα να προσδιορίζει ένα σύνολο από bytes για το οποίο θα γίνει αναζήτηση στα περιεχόμενα του πακέτου. Για παράδειγμα, αυτή η λειτουργία χρησιμοποιείται για την αναζήτηση οποιουδήποτε ASCII ή δυαδικού συνόλου bytes, το οποίο μπορεί να αποτελεί απειλή για ένα δίκτυο

Αναζήτηση ανώμαλου πακέτου Επιτρέπει σε ένα κανόνα να προσδιορίζει χαρακτηριστικά σε ένα πακέτο ή μια κεφαλίδα πακέτου τα οποία αποτελούν λόγο ανησυχίας, σε περίπτωση που είναι ασυνήθιστα ή ανώμαλα. Για παράδειγμα, μια τέτοια αναζήτηση επιτρέπει την αναζήτηση ενός ICMP πακέτου με μέγεθος μεγαλύτερο των 800 bytes. Η λειτουργία αυτή δηλάδή, δεν εκτελεί κάποιο είδος αναζήτησης, αλλά επικεντρώνεται στα χαρακτηριστικά του πακέτου

Η μηχανή επιθεώρησης πολλαπλών κανόνων χρησιμοποιεί μια τροποποιήσιμη υψηλής απόδοσης μηχανή αναζήτησης για να βρει όλες τις εμφανίσεις ενός πεδίου πρωτοκόλλου ή ενός περιεχομένου. Η αναζήτηση ανώμαλου πακέτου χρησιμοποιεί ένα σχήμα αναζήτησης το οποίο βασίζεται στον πρότυπο τρόπο επεξεργασίας κανόνων του Snort, αυτόν δηλαδή πριν την έκδοση 2.0. Όταν ένα πακέτο ταιριάζει με ένα κανόνα, το Snort επικυρώνει το κανόνα. Από τη στιγμή που επικυρώνεται ο κανόνας, δημιουργείται ένα συμβάν και προστίθεται στην ουρά των συμβάντων. Μόλις τελειώσει η επιθεώρηση του πακέτου, ξεκινά η επεξεργασία της ουράς συμβάντων από τον επιλογέα συμβάντων.

Η λειτουργία της υψηλής απόδοσης μηχανής αναζήτησης πραγματοποιείται σε δυο στάδια. Το πρώτο στάδιο είναι η εύρεση όλων των περιεχομένων προς αναζήτηση μέσα στο πακέτο και η ταυτοποίηση των κανόνων στους οποίους ανήκουν τα συγκεκριμένα περιεχόμενα. Στο δεύτερο στάδιο ελέγχεται αν και οι υπόλοιπες ρυθμίσεις του κανόνα ταιριάζουν στο πακέτο. Με αυτό τον τρόπο είτε επικυρώνεται ο κανόνας, δηλαδή και το περιεχόμενο βρέθηκε και οι υπόλοιπες ρυθμίσεις του κανόνα ταιριάζουν στο πακέτο, είτε βγαίνει το συμπέρασμα ότι παρόλο που βρέθηκε το περιεχόμενο οι υπόλοιπες ρυθμίσεις του κανόνα δεν επαληθεύτηκαν στο πακέτο.

Κατά τη διάρκεια του πρώτου σταδίου, η αναζήτηση των περιεχομένων γίνεται με τη βοήθεια κάποιων αλγόριθμων ομαδικής αναζήτησης. Οι αλγόριθμοι αυτοί στις τρέχουσες εκδόσεις του Snort είναι:

- μια τροποποίηση του αλγόριθμου Wu-Manber[23]
- ο αλγόριθμος Aho-Corasick[5]

- ο αλγοριθμος Boyer-Moore[16] για ομάδες μέχρι 10 κανόνων

3.5.3 Ο επιλογέας Συμβάντων

Η ουρά συμβάντων επιτρέπει στο Snort να ανιχνεύει κάθε ταίριασμα οποιουδήποτε κανόνα μέσα στο πακέτο. Ο ρόλος του επιλογέα Συμβάντων είναι να αναθέση μια προτεραιότητα σε αυτά τα συμβάντα και να επιλέξει τα συμβάντα από την ουρά ανάλογα με την προτεραιότητα που τους ανατέθηκε. Προς το παρών, το συμβάν με το μεγαλύτερο μήκος αντιστοιχίας θεωρείται ότι έχει την μεγαλύτερη προτεραιότητα και επιλέγεται. Το συμβάν αυτό αποστέλλεται στο σύστημα εξόδου του Snort.

Κεφάλαιο 4

Περιγραφή της Υλοποίησης

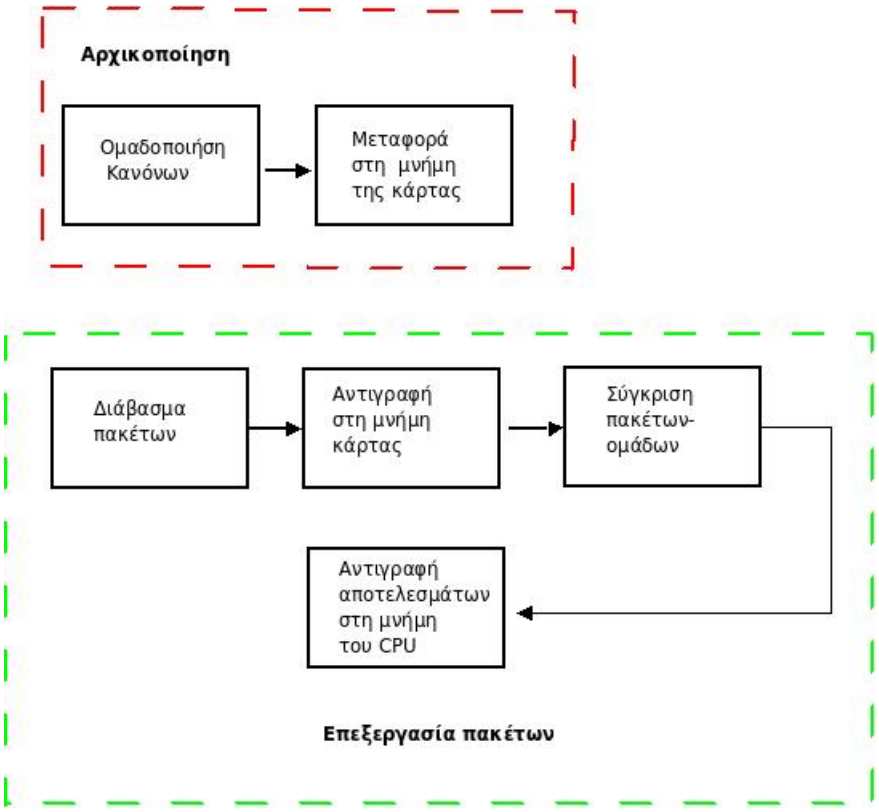
Όπως αναφέρθηκε και στην εισαγωγή, σκοπός της εργασίας αυτής είναι η υλοποίηση της κατηγοριοποίησης επικεφαλίδας διερχόμενων πακέτων με τις δυνατότητες που παρέχει μια σύγχρονη κάρτα γραφικών. Οι ενέργειες που εκτελεί το σύστημα για την υλοποίηση του σκοπού φαίνονται στο σχήμα 4.1.

Όπως φαίνεται και στο σχήμα 4.1, η λειτουργία του συστήματος γίνεται σε δύο φάσεις, την αρχικοποίηση και την επεξεργασία των πακέτων. Σε αφαιρετικό επίπεδο, κατά τη φάση της αρχικοποίησης πραγματοποιείται η φόρτωση των κανόνων στην κάρτα γραφικών, ενώ κατά τη φάση της επεξεργασίας πακέτων πραγματοποιείται η σύγκριση των πακέτων με τους κανόνες. Οι λεπτομέρειες των μπλοκ που φαίνονται στο σχήμα 4.1, θα περιγραφούν στη συνέχεια του κεφαλαίου.

Σε επίπεδο υλικού, το σύστημα που υλοποιήθηκε αποτελείται από δυο μέρη, τον κύριο επεξεργαστή του υπολογιστή και την κάρτα γραφικών. Ο μόνος τρόπος επικοινωνίας μεταξύ του επεξεργαστή και της κάρτας γραφικών είναι διαμέσου της global μνήμης της κάρτας γραφικών, όπως φαίνεται και στο σχήμα 4.2.

Προκειμένου να περιγραφεί η λειτουργία του κάθε μπλοκ του σχήματος 4.1, πρέπει να γίνει σαφές πιο μέρος του υλικού (ο επεξεργαστής ή η κάρτα γραφικών) εκτελεί το κάθε μπλοκ. Αυτό φαίνεται στον πίνακα 4.1. Στον προηγούμενο πίνακα, κάθε κελί που σημειώνεται με ✓, δηλώνει ότι η συγκεκριμένη ενέργεια της γραμμής εκτελείται από το στοιχείο της αντίστοιχης στήλης. Αντίστοιχα, η παύλα δηλώνει ότι η συγκεκριμένη ενέργεια της γραμμής δεν εκτελείται από το στοιχείο της αντίστοιχης στήλης.

Όσον αφορά στα νήματα, επειδή υπάρχουν δύο συστατικά (ο επεξεργαστής και η κάρτα γραφικών) στο σχήμα 4.3 φαίνεται ότι υπάρχει ένα νήμα στον επεξεργαστή, το οποίο είναι υπεύθυνο για εκτέλεση των λειτουργιών που κάνει ο επεξεργαστής αλλά και για την έναρξη της εκτέλεσης της κάρτας γραφικών. Να σημειωθεί ότι το σχήμα προσπαθεί να δώσει μια γενική ιδέα για την ροή

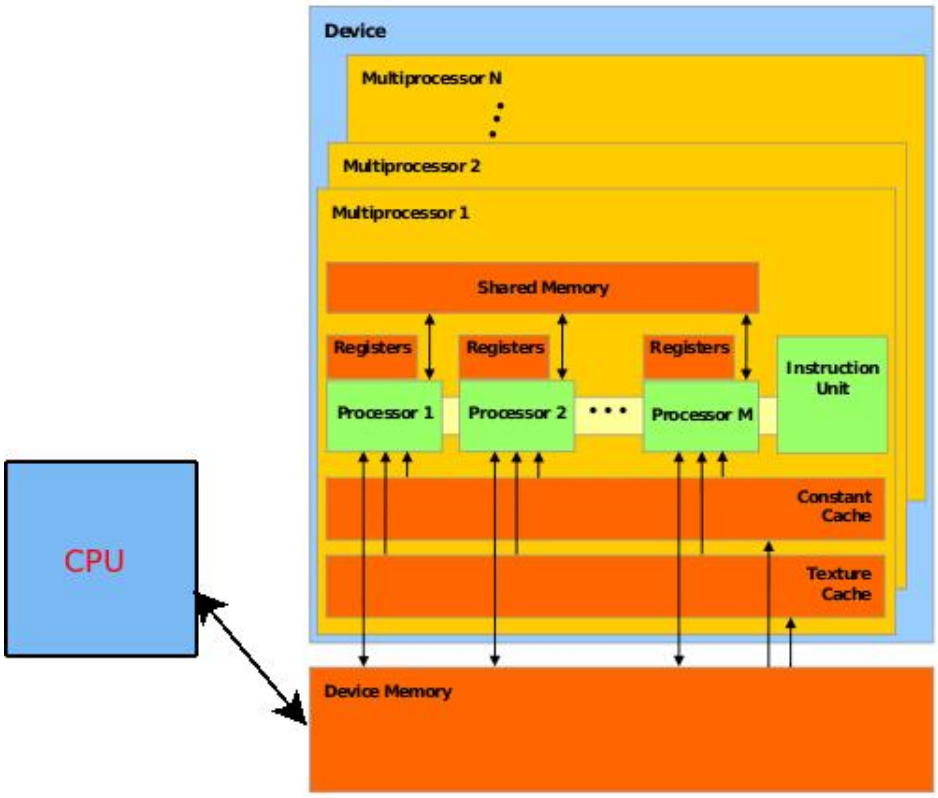


Σχήμα 4.1: Διάγραμμα μπλοκ της αρχιτεκτονικής

Πίνακας 4.1: Οι ενέργειες που εκτελούν ο επεξεργαστής και κάρτα γραφικών

	CPU	**Κάρτα Γραφικών**
Ομαδοποίηση κανόνων	✓	-
Μεταφορά στη μνήμη της κάρτας	✓	-
Διάβασμα πακέτων	✓	-
Αντιγραφή στη μνήμη της κάρτας	✓	-
Σύγκριση πακέτων ομάδων	-	✓
Αντιγρφή αποτελεσμάτων στη μνήμη του CPU	✓	-

εκτέλεσης του προγράματος. Για παράδειγμα, στο κομμάτι που αναφέρεται στην κάρτα γραφικών (device) το πλήθος των νημάτων που εκτελούνται στην πραγματικότητα είναι μεγαλύτερο από αυτό που φαίνεται στο σχήμα.



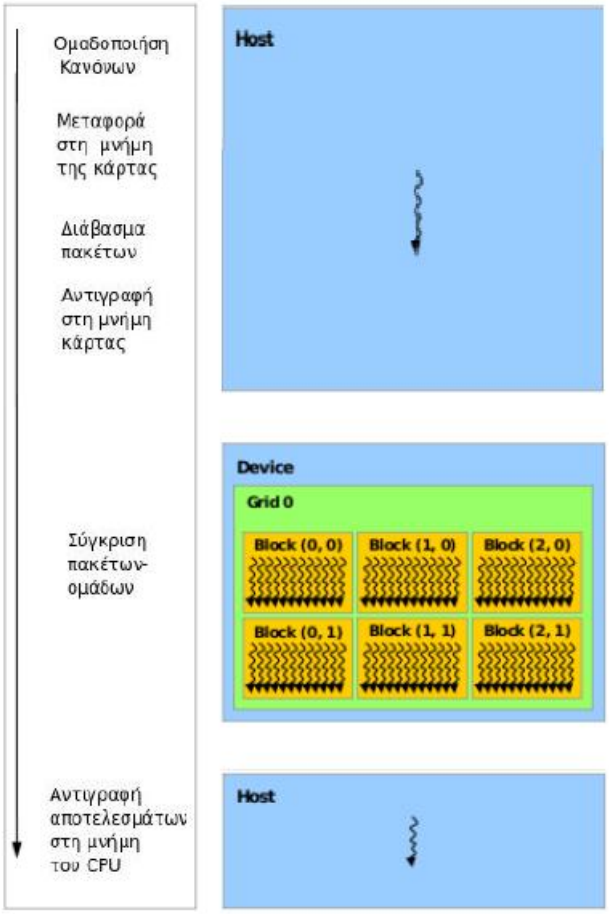
Σχήμα 4.2: Μακροσκοπική εικόνα του συστήματος που υλοποιήθηκε

Στη συνέχεια του κεφαλαίου θα περιγραφεί αναλυτικά το κάθε συστατικό του σχήματος 4.1. Στην συνέχεια του κειμένου, όπου αναφέρεται host εννοείται ο κύριος επεξεργαστής του υπολογιστή και σαν device ορίζεται η κάρτα γραφικών, όπως άλλωστε αναφέρθηκε και στο κεφάλαιο 2.

4.1 Ομαδοποίηση των κανόνων του Snort

Στο στάδιο αυτό πραγματοποιήθηκε η ομαδοποίηση των κανόνων του Snort. Ο λόγος που ομαδοποιούνται οι κανόνες, όπως αναφέρεται και στο Κεφάλαιο 3, είναι ότι λόγω μεγάλου πλήθους των κανόνων δεν θα ήταν αποδοτικό να γίνεται η επεξεργασία του κάθε κανόνα ξεχωριστά. Έτσι, δημιουργούνται ομάδες από κανόνες με κοινά χαρακτηριστικά. Στο σημείο αυτό να τονιστεί ότι η ομαδοποίηση δεν έγινε ακριβώς όπως γίνεται στο Snort, αλλά με βάση την διπλωματική του Κ. Δημόπουλου [27].

Ειδικότερα, χρησιμοποιήθηκαν τα αρχεία κανόνων του snort της έκδοσης



Σχήμα 4.3: Διαχωρισμός του νήματος του επεξεργαστή από τα νήματα της κάρτας γραφικών

2.8.3.2. Η ομαδοποίηση των κανόνων έγινε με βάση τις παραμέτρους των επικεφαλίδων των κανόνων. Όπως έχει αναφερθεί, ένας κανόνας του snort αποτελείται από δυο λογικά μέρη, την επικεφαλίδα και τις ρυθμίσεις του κανόνα. Από την επικεφαλίδα του κανόνα λήφθηκαν υπόψη όλα τα συστατικά της, εκτός από την ενέργεια του κανόνα. Δηλαδή, στην ομαδοποίηση δεν λήφθηκε υπόψη αν κάποιος κανόνας ήταν τύπου alert, log, pass, activate ή dynamic. Συνεπώς, έγινε ομαδοποίηση των κανόνων με βάση την διεύθυνση IP του αποστολέα και του παραλήπτη, την πόρτα του αποστολέα και του παραλήπτη, το πρωτόκολλο του κανόνα και τον τελεστή κατεύθυνσης.

Η ομαδοποίηση έγινε σε δυο φάσεις. Αρχικά οι κανόνες ομαδοποιούνται με βάση το πρωτόκολλο στο οποίο αναφέρονται. Γι' αυτό το λόγο δημιουργούνται

τέσσερις πίνακες, αρχικά κενοί, ένας για κάθε πρωτόκολλο. Στη συνέχεια, οι κανόνες ομαδοποιούνται με βάση την τετράδα: *διεύθυνση IP αποστολέα*, *πόρτα αποστολέα*,*διεύθυνση IP παραλήπτη*, *πόρτα παραλήπτη* για τα πρωτόκολλα TCP/UDP ή με βάση το ζευγάρι *διεύθυνση IP αποστολέα*,*διεύθυνση IP παραλήπτη* για τα πρωτόκολλα IP και ICMP. Πιο συγκεκριμένα, η ομαδοποίηση γίνεται ως εξής:

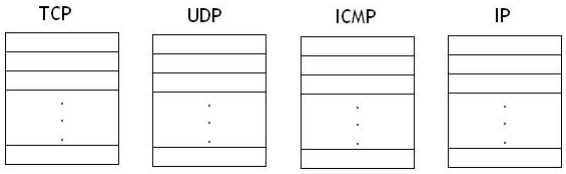
1. Διάβασμα ενός κανόνα από τα αρχεία κανόνων του snort
2. Επιλογή του πίνακα στον οποίο θα αποθηκευτεί ο κανόνας, ανάλογα με το πρωτόκολλο του κανόνα
3. Σύγκριση της τετράδας ή του ζευγαριού των παραμέτρων του κανόνα με τις αντίστοιχες παραμέτρους των ομάδων του πίνακα που επιλέχθηκε. Σε περίπτωση ύπαρξης ομάδας με τις ίδιες παραμέτρους, ο κανόνας αποθηκεύεται στη συγκεκριμένη ομάδα. Διαφορετικά, δημιουργείται νέα ομάδα με παραμέτρους τις παραμέτρους του κανόνα

Η παραπάνω διαδικασία επαναλαμβάνεται για όλους τους κανόνες. Ένα παράδειγμα θα αποσαφηνίσει την προηγούμενη διαδικασία. Έστω ότι ένα αρχείο κανόνων του Snort περιέχει τους παρακάτω κανόνες:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:foo1;sid:1000;)
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:foo2;sid:1001;)
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:foo3;sid:1002;)
alert udp $HOME_NET 53 -> $EXTERNAL_NET 100 (msg:foo4;sid:1003;)
alert ip any any -> any any (msg:foo5;sid:1004;)
alert icmp $HOME_NET any -> $EXTERNAL_NET any (msg:foo6;sid:1005;)
alert tcp 147.27.3.0/24 any -> $HOME_NET 80:100 (msg:foo7;sid:2000;)
```

Τότε, πριν και μετά την ομαδοποίηση οι πίνακες στην μνήμη φαίνονται στο σχήμα 4.4.

Στο σχήμα 4.4(b) φαίνονται οι ομάδες που δημιουργήθηκαν και ο τρόπος που είναι αποθηκευμένες στην μνήμη. Η στήλη με την ένδειξη rules αναφέρεται στα sid των κανόνων που περιέχονται στην εκάστοτε ομάδα. Τα $HOME_NET και $EXTERNAL_NET είναι συμβολικά ονόματα για διευθύνσεις IP και δηλώνονται στο configuration file του Snort. Η χρησιμότητά τους έγκειται στο ότι παρέχουν στον χρήστη του Snort έναν εύκολο τρόπο να δηλώσουν στο Snort το δίκτυο προς προστασία. Υπενθυμίζεται από το κεφάλαιο 3 ότι ο συμβολισμός 80:100 στον κανόνα με sid:2000 δηλώνει εύρος πόρτας, με το 80 να είναι το κάτω όριο και το 100 το άνω όριο. Σημειώνεται ότι αν στο προηγούμενο παράδειγμα υπήρχε ένας κανόνας που είχε στην πόρτα προορισμού



(a) Πριν την ομαδοποίηση

TCP

Παράμετροι ομάδας κανόνων				
srcIp	srcPort	dstIp	dstPort	rules
$home_net	any	$external_net	any	1000
$home_net	any	$external_net	80	1001, 1002
147.27.3.0/24	any	$home_net	80:100	2000

UDP

Παράμετροι ομάδας κανόνων				
srcIp	srcPort	dstIp	dstPort	rules
$external_net	53	$home_net	100	1003

ICMP

Παράμετροι ομάδας κανόνων		
srcIp	dstIp	rules
any	any	1004

IP

Παράμετροι ομάδας κανόνων		
srcIp	dstIp	rules
$home_net	$external_net	1005

(b) Μετά την ομαδοποίηση

Σχήμα 4.4: Η ομαδοποίηση των κανόνων στην πράξη

μια τιμή ανάμεσα στο 80 και στο 100, δεν θα άνηκε στην ίδια ομάδα με τον κανόνα 2000. Ο λόγος είναι οτι δεν γίνεται κάποιου είδους βελτιστοποίηση των ομάδων.

Ειδική περίπτωση αποτελούν οι αμφίδρομοι κανόνες, οι κανόνες δηλάδη που έχουν τον τελεστή <>. Στην περίπτωση αυτή, λόγω του ότι δεν έχει σημασία η κατεύθυνση της κίνησης, δημιουργούνται δύο κανόνες, ένας για κάθε κατεύθυνση και ακολουθείται η παραπάνω διαδικασία μια φορά για κάθε κανόνα που προέκυψε. Για παράδειγμα, ο κανόνας 1 θα δημιουργήσει τους κανόνες 2 και 3:

```
alert udp $HOME_NET 100 <> $EXTERNAL_NET 53 (msg:foo;sid:3000) (1)
alert udp $HOME_NET 100 -> $EXTERNAL_NET 53 (msg:foo;sid:3000) (2)
```

```
alert udp $EXTERNAL_NET 53 -> $HOME_NET 100 (msg:foo;sid:3000) (3)
```

Συνεχίζοντας το παράδειγμα του σχήματος 4.4, αν προστεθεί και ο κανόνας (1), τότε στον πίνακα για το πρωτόκολλο UDP, ο κανόνας (2) θα αποτελέσει μόνος του ομάδα, ενώ ο κανόνας (3) θα εισαχθεί στην ομάδα του κανόνα με sid:1003, όπως φαίνεται και στο σχήμα 4.5.

TCP

Παράμετροι ομάδας κανόνων				
srcIp	srcPort	dstIp	dstPort	rules
$home_net	any	$external_net	any	1000
$home_net	any	$external_net	80	1001, 1002
147.27.3.0/24	any	$home_net	80:100	2000

UDP

Παράμετροι ομάδας κανόνων				
srcIp	srcPort	dstIp	dstPort	rules
$external_net	53	$home_net	100	1003, 3000
$home_net	100	$external_net	53	3000

ICMP

Παράμετροι ομάδας κανόνων		
srcIp	dstIp	rules
any	any	1004

IP

Παράμετροι ομάδας κανόνων		
srcIp	dstIp	rules
$home_net	$external_net	1005

Σχήμα 4.5: Παράδειγμα αμφίδρομων κανόνων

Το σύνολο κανόνων που χρησιμοποιήθηκε περιείχε 9292 κανόνες, κατανεμημένων όπως φαίνεται στο σχήμα 4.2. Με τη παραπάνω διαδικασία παρήχθησαν, 484 ομάδες στο TCP, 104 στο UDP, 3 στο IP και 4 στο ICMP, όπως φαίνεται και στη τελευταία στήλη του πίνακα 4.2.

Στο σημείο αυτό, να υπενθυμιστεί από το πίνακα 4.1 ότι η διαδικασία της ομαδοποίησης λαμβάνει χώρα εξ'ολοκλήρου στον host, στον κύριο επεξεργαστή του συστήματος. Επομένως, οι παραπάνω πίνακες βρίσκονται στην RAM του υπολογιστή και πρέπει να αντιγραφούν στην μνήμη της κάρτας, καθώς έτσι θα είναι προσπελάσιμοι και από την κάρτα.

Πίνακας 4.2: Κατανομή κανόνων στο Snort και ομάδων που προέκυψαν μετά τη κατηγοριοποίηση

Πρωτόκολλο	**Πλήθος κανόνων**	**Πλήθος Ομάδων μετα τη κατηγοριοποίηση**
TCP	8496	484
UDP	631	104
ICMP	131	4
IP	34	3
Σύνολο	9292	595

4.2 Μεταφορά των παραγόμενων ομάδων στην μνήμη της κάρτας

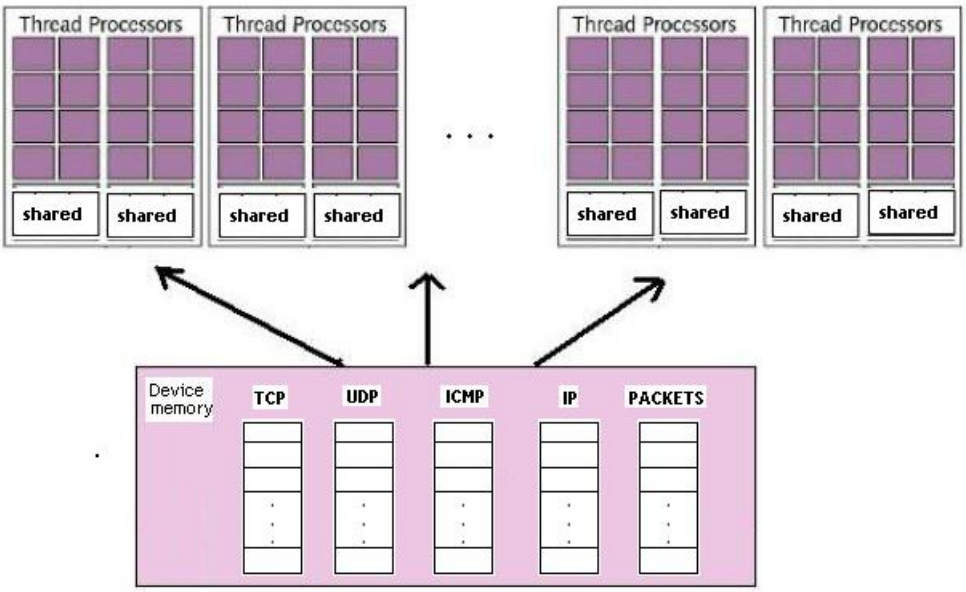
Μόλις ολοκληρωθεί η ομαδοποίηση, ο host αντιγράφει τους παραγόμενους πίνακες των ομάδων στην global μνήμη της κάρτας γραφικών με χρήση του API που παρέχει η CUDA και πιο συγκεκριμένα με την συνάρτηση cudaMemcpy(). Επειδή υπάρχουν τέσσερις πίνακες που πρέπει να αντιγραφούν, η συνάρτηση cudaMemcpy() καλείται τέσσερις φορές, μια για κάθε πίνακα. Η συνάρτηση αυτή είναι σύγχρονη, υπό την έννοια ότι η εκτέλεσή της δεν συνεχίζεται αν δεν ολοκληρωθεί η αντιγραφή. Δηλαδή, το νήμα του host που καλεί την συνάρτηση cudaMemcpy() περιμένει μέχρι να τερματίσει η συνάρτηση.

4.3 Μεταφορά των πακέτων στην κάρτα γραφικών

Η μεταφορά των πακέτων από την μνήμη του host στη μνήμη του device, γίνεται με παρόμοιο τρόπο, όπως με τη μεταφορά των ομάδων. Το πως αποκτά τα πακέτα ο host περιγράφεται στην παράγραφο 5.2.1. Προς το παρόν μπορεί να θεωρηθεί ότι στη μνήμη του host βρίσκεται ένας πίνακας με τις επικεφαλίδες των πακέτων. Οπότε, η αντιγραφή αυτού του πίνακα από τη μνήμη του host στη μνήμη του device, γίνεται πάλι με τη συνάρτηση cudaMemcpy().

4.4 Σύγκριση πακέτων-κανόνων στην κάρτα γραφικών

Αφού ολοκληρωθεί η ομαδοποίηση των κανόνων, η αντιγραφή τους στη global μνήμη της κάρτας και η αντιγραφή των πακέτων, το επόμενο στάδιο είναι, η σύγκριση των παραμέτρων της κεφαλίδας των πακέτων με τις παραμέτρους των ομάδων που προέκυψαν. Επειδή ο στόχος ήταν η σύγκριση αυτή να γίνεται παράλληλα, το στάδιο αυτό λαμβάνει χώρα εξ'ολοκλήρου στη κάρτα γραφικών με χρήση του μοντέλου της CUDA, δηλαδή υλοποιείται με τη χρήση ενός kernel, το οποίο εκτελείται από πολλά νήματα παράλληλα. Στο σχήμα 4.6 φαίνεται η κατάσταση στην global μνήμη της κάρτας γραφικών. Μέχρι αυτό το σημείο, στην μνήμη της κάρτας περιέχονται οι τέσσερις πίνακες με τις παραμέτρους των ομάδων και οι επικεφαλίδες των πακέτων. Στο πάνω μέρος του σχήματος φαίνονται οι πολυεπεξεργαστές της κάρτας.



Σχήμα 4.6: Κατάσταση στην κάρτα γραφικών

Να υπενθυμιστεί στο σημείο αυτό από το κεφάλαιο 2 ότι, πρόσβαση στην global μνήμη της κάρτας έχουν όλοι οι πολυεπεξεργαστές. Επειδή όπως αναφέρθηκε η σύγκριση των πακέτων με τις ομάδες των κανόνων θα γίνει παράλληλα, με τη χρήση ενός kernel, πρέπει να διασαφηνιστούν ορισμένα πράγματα όπως, το πλήθος των νημάτων που θα εκτελέσουν το kernel, η οργάνωση των νημάτων αυτών, ο τρόπος προσπέλασης των ομάδων και πακέτων από τους

πολυεπεξεργαστές, ο τρόπος με τον οποίο πραγματοποιείται η σύγκριση και γράφονται τα αποτελέσματα. Όλα αυτά θα περιγραφούν στις παραγράφους που ακολουθούν.

4.4.1 Επίπεδο παραλληλισμού που επιλέχθηκε

Το επίπεδο παραλληλισμού που επιλέχθηκε, αφορούσε τις ομάδες. Δηλαδή, η κάρτα γραφικών συγκρίνει ένα πακέτο τη φορά με όλες τις ομάδες παράλληλα. Ο λόγος για τον οποίο επιλέχθηκε παραλληλισμός σε επίπεδο ομάδας έχει να κάνει με την φιλοσοφία του Snort, σύμφωνα με την οποία ο βελτιστοποιητής του Snort 3.5.1 προσπαθεί να δημιουργήσει ομάδες οι οποίες θα μπορούν να συγκριθούν παράλληλα.

4.4.2 Οργάνωση των νημάτων

Από τα πρώτα πράγματα που πρέπει να αποφασιστούν κατά την σχεδίαση ενός kernel είναι το πλήθος και η οργάνωση των νημάτων που θα το εκτελούν. Υπενθυμίζεται ότι, τα νήματα που εκτελούν ένα kernel, οργανώνονται σε ομάδες ίδιου μεγέθους, τα μπλόκ, τα οποία εκτελούνται στον ίδιο πολυεπεξεργαστή (κεφάλαιο 2). Γενικά το πλήθος των νημάτων εξαρτάται από το πλήθος των δεδομένων που επεξεργάζονται. Στο συγκεκριμένο πρόβλημα, επειδή η πλειοψηφία των ομάδων που προέκυψαν αναφέρεται στο TCP, επιλέχθηκε το πλήθος των νημάτων να είναι ίσο με το πλήθος των ομάδων στον πίνακα TCP, το οποίο είναι 484, και όχι ίσο με το πλήθος όλων των ομάδων συνολικά. Για λόγους που έχουν να κάνουν με την απόδοση της κάρτας [9, 10], ο αριθμός των νημάτων ενός μπλόκ πρέπει να είναι πολλαπλάσιο του μεγέθους του warp, και επίσης το πλήθος των μπλοκ πρέπει να είναι τουλάχιστον όσο το πλήθος των πολυεπεξεργαστών. Ιδανικά το πλήθος των μπλοκ πρέπει να είναι πολύ μεγαλύτερο από το πλήθος των πολυεπεξεργαστών. Έτσι, επιλέχθηκε το μέγεθος του μπλόκ να είναι 32 νήματα, όσο είναι και το μέγεθος του warp στην συγκεκριμένη κάρτα. Το πλήθος των μπλόκ τότε είναι, $\lceil \frac{μγεθοςTCP}{μγεθοςμπλκ} \rceil + 1 = 16$. Με αυτόν τον τρόπο, το πλήθος των νημάτων είναι $32 * 16 = 512$.

Σε κάθε ένα νήμα ανατίθεται ένα αναγνωριστικό το οποίο προκύπτει ώς εξής: $idx = blockIdx.x * blockDim.x + threadIdx.x$, όπου

- blockIdx.x, το αναγνωριστικό του μπλόκ μέσα στο πλέγμα(grid)
- blockDim.x, το μέγεθος του μπλόκ σε νήματα
- threadIdx.x, το αναγνωριστικό ενός νήματος μέσα στο μπλόκ

Έτσι για παράδειγμα, για το μπλόκ 0 το idx θα πάρει τίμες 0,...,31, ενώ για το μπλόκ 1 το idx θα πάρει τιμές 32,..,63 και ου το καθ' εξής. Συνολικά η idx θα πάρει τιμές 0 έως 511.

4.4.3 Κύρια λειτουργία του kernel

Μέχρι αυτό το σημείο, οι πίνακες με τις ομάδες των κανόνων, βρίσκονται στην global μνήμη της κάρτας γραφικών. Όπως έχει περιγραφεί στο κεφάλαιο 2, η global μνήμη είναι η πιο αργή από τις μνήμες στην ιεραρχία μνήμης της CUDA. Επιπλέον, η ανάγνωση από αυτή τη μνήμη δε θα γίνει μόνο μια φορά, αλλά μια φορά για κάθε πακέτο. Επομένως, μια καλύτερη λύση είναι η τοποθέτηση των πινάκων αυτών σε μια από τις υπόλοιπες μνήμες της ιεραρχίας μνήμης. Η μνήμη που επιλέχθηκε για αυτό το σκοπό είναι η κοινόχρηστη μνήμη του κάθε πολυεπεξεργαστή. Επομένως, η πρώτη λειτουργία του kernel είναι να αντιγράψει τους πίνακες από τη global μνήμη στη κοινόχρηστη μνήμη κάθε πολυεπεξεργαστή. Να σημειωθεί στο σημείο αυτό ότι οι κοινόχρηστες μνήμες των πολυεπεξεργαστών είναι προσβάσιμες μόνο από το device και όχι από το host. Επομένως, η διαδικασία της αντιγραφής των ομάδων από την global μνήμη στις κοινόχρηστες μνήμες πραγματοποιείται από την κάρτα γραφικών και θα περιγραφεί αναλυτικά στη συνέχεια του κεφαλαίου. Περιληπτικά, κάθε νήμα θα αντιγράψει την ομάδα που του αντιστοιχεί.

Αφού ολοκληρωθεί η αντιγραφή στις κοινόχρηστες μνήμες, η επόμενη ενέργεια είναι η διαδικασία της σύγκρισης των πακέτων με τις ομάδες που βρίσκονται ήδη στις κοινόχρηστες μνήμες. Η διαδικασία πραγματοποιείται μέσα σε ένα βρόγχο και σε κάθε επανάληψη του βρόγχου διαβάζεται ένα πακέτο. Ανάλογα με το πρωτόκολλο του πακέτου επιλέγεται ο κατάλληλος πίνακας ομάδων. Αφού επιλεχθεί ο πίνακας, όλα τα νήματα συγκρίνουν το πακέτο με όλες τις ομάδες του πίνακα που επιλέχθηκε. Αφού ολοκληρωθεί η σύγκριση, γράφονται τα αποτελέσματα σε ένα πίνακα στην global μνήμη και η διαδικασία συνεχίζεται με το επόμενο πακέτο. Το κάθε κομμάτι της προηγούμενης διαδικασίας θα περιγραφεί αναλυτικά στις επόμενες παραγράφους. Στο παρακάτω κομμάτι ψευδοκώδικα φαίνεται η λειτουργία του kernel.

Όπως φαίνεται, η λειτουργία του kernel χωρίζεται σε 3 στάδια, την αντιγραφή των ομάδων στις κοινόχρηστες μνήμες, τη διαδικασία της σύγκρισης και την εγγραφή των αποτελεσμάτων.

```
__global__ void foo(...)
{
        ...
        /*load from global to shared memory*/
        shared_tcp=load(global_tcp);
```

```
        shared_udp=load(global_udp);
        shared_ip=load(global_ip);
        shared_icmp=load(global_icmp);
        __syncthreads();
        for(i=0;i<NUMBER_OF_PACKETS;i++){
           p=read_Packet();
           switch(p.protocol){
                case TCP: compare(p,shared_tcp);
                case UDP: compare(p,shared_udp);
                case ICMP: compare(p,shared_icmp);
                case IP: compare(p,shared_ip);
           }
           /*write results to global memory*/
           write_result();
        }
}
```

Αντιγραφή ομάδων στις κοινόχρηστες μνήμες

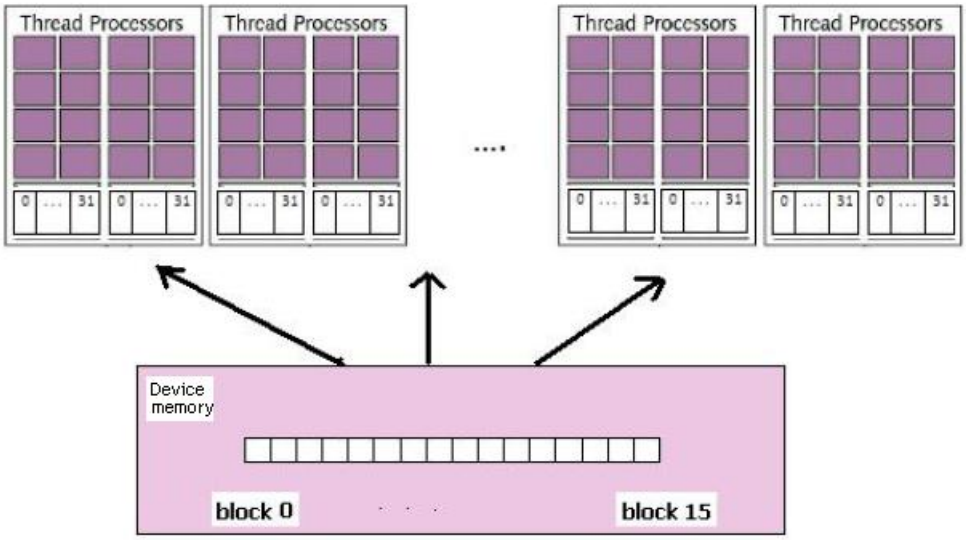
Όπως αναφέρθηκε και στην περιγραφή του μοντέλου της CUDA, η global μνήμη της κάρτας γραφικων δεν είναι εναποθηκευμένη (cached). Επιπλέον, έχει το χαμηλότερο εύρος ζώνης σε σχέση με τους υπόλοιπους χώρους μνήμης της κάρτας. Γι' αυτό το λόγο, η πρώτη ενέργεια που κάνει το kernel, είναι να αντιγράψει τους πίνακες με τις ομάδες κανόνων, από τη global μνήμη στη κοινόχρηστη μνήμη κάθε πολυεπεξεργαστή.

Ο τρόπος που γίνεται η αντιγραφή είναι ο εξής: κάθε μπλόκ αντιστοιχεί σε ένα κομμάτι του πίνακα των ομάδων και είναι υπεύθυνο για την αντιγραφή του κομματιού που του αντιστοιχεί. Κάθε νήμα του μπλοκ αντιγράφει 1 θέση του πίνακα σε ένα πίνακα 32 θέσεων που βρίσκεται στις κοινόχρηστες μνήμες κάθε πολυεπεξεργαστή. Επειδή η κάρτα έχει 16 πολυεπεξεργαστές υπάρχουν 16 τέτοιοι πίνακες, στους οποίους αποθηκεύονται οι παράμετροι των ομάδων. Ο τρόπος που γίνεται το indexing μεταξύ των 2 μνημών, υποθέτωντας ότι ο πίνακας shared βρίσκεται στην κοινόχρηστη μνήμη και ο πίνακας global στην global μνήμη, είναι:

```
int tid=threadIdx.x;
__shared__ shared[32];
shared[tid]=global[idx]
```

Με αυτόν τον τρόπο, κάθε μπλόκ αντιγράφει το κομμάτι του και το i-στο νήμα ενός μπλοκ αντιγράφει την idx-στη θέση του πίνακα στην i-οστη θέση

του πίνακα της κοινόχηστης μνήμης. Για παράδειγμα, το νήμα με idx 80 έχει blockIdx.x=2 και tid=16 θα αντιγράψει την θέση 80 του πίνακα που βρίσκεται στη global μνήμη, στη θέση 16 του πίνακα που βρίσκεται στη κοινόχρηστη μνήμη κάποιων από τους πολυεπεξεργαστές και πιο συγκεκριμένα του πολυεπεξεργαστή που θα εκτελέσει το μπλοκ 2. Να σημειωθεί ότι οι πίνακες που βρίσκονται στις κοινόχρηστες μνήμες, έχουν μέγεθος όσο και το μέγεθος του μπλοκ και δεν χρειάζεται να έχουν μεγαλύτερο από αυτό. Στην ουσία η παραπάνω διαδικασία αντιγραφής λαμβάνει χώρα 4 φορές, μια φορά για κάθε πίνακα στην global μνήμη. Η διαδικασία της αντιγραφής φαίνεται στο σχήμα 4.7



Σχήμα 4.7: Αντιγραφή στις κοινόχρηστες μνήμες

Μόλις ολοκληρωθεί η αντιγραφή, εκτελείται η built-in εντολή της CUDA __syncthreads(). Η συνάρτηση αυτή, λειτουργεί σαν μπάρα συγχρονισμού (barrier) και κανένα νήμα στο μπλοκ δεν συνεχίζει την εκτέλεση των επόμενων εντολών, μέχρι να ολοκληρωθεί η αντιγραφή.

Η διαδικασία της σύγκρισης

Αφού γίνει η αντιγραφή, γίνεται η σύγκριση των πακέτων με τις ομάδες των κανόνων. Προκειμένου να περιγραφεί η διαδικασία της σύγκρισης, πρέπει πρώτα να περιγραφεί η μοντελοποίηση μιας ομάδας κανόνων και η μοντελοποίηση της κεφαλίδας πακέτων. Η μοντελοποίηση μιας ομάδας κανόνων έγινε ώς εξής:

- unsigned int srcIp, η διεύθυνση IP πηγής για την ομάδα

- unsigned int dstIp, η διεύθυνση IP προορισμού για την ομάδα
- unsigned int srcMask, η μάσκα δικτύου για την διεύθυνση IP πηγής
- unsigned int dstMask, η μάσκα δικτύου για την διεύθυνση IP προορισμού
- unsigned short int srcLow, το κάτω όριο για την πόρτα του αποστολέα της ομάδας
- unsigned short int srcHigh, το άνω όριο για την πόρτα του αποστολέα της ομάδας
- unsigned short int dstLow, το κάτω όριο για την πόρτα του παραλήπτη της ομάδας
- unsigned short int dstHigh, το άνω όριο για την πόρτα του παραλήπτη της ομάδας
- char invert, ένα byte που κρατάει όλη την απαραίτητη πληροφορία για το αν τα προηγούμενα πεδία είναι ανεστραμένα ή όχι

Σύμφωνα με το κεφάλαιο 3, μια διεύθυνση IP στο snort μπορεί να δηλωθεί είτε ως διεύθυνση ενός συγκεκριμένου μηχανήματος (π.χ 147.27.39.100), είτε να συνδυάζεται και με μια μάσκα δικτύου (π.χ 147.27.39.0/24), είτε με τη λέξη κλειδί any. Γι' αυτό το λόγο χρησιμοποιήθηκε μια λέξη για την διεύθυνση IP και μια για την μάσκα δικτύου. Να σημειωθεί ότι υλοποιήθηκαν μόνο διευθύνσεις κλάσης A, B, C και D. Η λέξη κλειδί any υλοποιήθηκε θέτωντας το πεδίο της διεύθυνσης αλλά και της μάσκας με μηδέν.

Όσον αφορά στις πόρτες, το snort (κεφάλαιο 3) επιτρέπει να ορίζονται απλές πόρτες(π.χ 80), εύρος πόρτας(π.χ 30:40) ή με τη λέξη κλειδί any. Γι' αυτό το λόγο χρησιμοποιήθηκαν, δύο 16-bit ποσότητες για το κάτω και το άνω όριο αντίστοιχα. Σε περίπτωση συγκεκριμένης πόρτας, τα δύο όρια είναι ίδια. Στην περίπτωση της λέξης κλειδί any, το κάτω όριο παίρνει τιμή μηδέν και το άνω 65536.

Το πεδίο invert αποτελεί ένα byte, το οποίο περιέχει πληροφορία σχετικά με τον τελεστή της αντιστροφής (!). Υπενθυμίζεται ότι το snort (κεφάλαιο 3) επιτρέπει τον τελεστή της αντιστροφής στην δήλωση των διευθύσεων IP αλλά και των πορτών. Συγκεκριμένα, στο byte αυτό κρατείται πληροφορία για 4 τελεστές αντιστροφής, των διευθύνσεων IP του αποστολέα και του παραλήπτη και των πορτών του αποστολέα και του παραλήπτη. Η πληροφορία που χρειάζεται για κάθε τελεστή είναι ένα bit, με το μηδέν να δηλώνει ότι δεν έχουμε αντιστροφή και το ένα ότι έχουμε. Κατά σύμβαση, τα 4 αυτά bit βρίσκονται στα 4 λιγότερα σημαντικά bit του invert και τα υπόλοιπα bit είναι μηδέν.

Έτσι το byte invert μπορεί να πάρει 16 διαφορετικές τιμές. Ο τρόπος με τον οποίο εξάγονται τα συγκεκριμένα bits είναι με λίγες δυαδικές πράξεις, όπως φαίνεται στον πίνακα 4.3

Πίνακας 4.3: Τρόπος εξαγωγής των bits αντιστροφής

b_3	$b_7b_6b_5b_4b_3b_2b_1b_0 >> 3$
b_2	$(b_7b_6b_5b_4b_3b_2b_1b_0\&0100) >> 2$
b_1	$(b_7b_6b_5b_4b_3b_2b_1b_0\&0010) >> 1$
b_0	$b_7b_6b_5b_4b_3b_2b_1b_0\&0001$

Η μοντελοποίηση της κεφαλίδας πακέτων έγινε ως εξής:

- unsigned int source_ip, η διεύθυνση IP αποστολέα του πακέτου
- unsigned int dest_ip, η διεύθυνση IP παραλήπτη του πακέτου
- unsigned int source_port, η πόρτα αποστολέα του πακέτου
- unsigned int dest_port, η πόρτα παραλήπτη του πακέτου
- char protocol, το πρωτόκολλο στο οποίο αναφέρεται τοπακέτο

Αφού περιγράφηκαν τα πεδία από τα οποία αποτελείται μια ομάδα κανόνων και η κεφαλίδα των πακέτων, θα περιγραφεί η διαδικασία της σύγκρισης.

Η σύγκριση πραγματοποιείται από κάθε νήμα. Κάθε νήμα συγκρίνει, δηλαδή, τις παραμέτρους της ομάδας που του αντιστοιχεί, με τις παραμέτρους ενός πακέτου. Προκειμένου να μην υπάρχουν αποκλίνοντα νήματα(κεφάλαιο 2) και να δημιουργούνται προβλήματα απόδοσης, η σύγκριση των παραμέτρων δεν γίνεται με συνθήκες (branches) αλλά με δυαδικές και λογικές πράξεις. Η σύγκριση πραγματοποιείται επαναληπτικά για όλα τα πακέτα. Δηλαδή, για κάθε επανάληψη του βρόγχου όλα τα νήματα σύγκρίνουν παράλληλα το τρέχων πακέτο με όλες τις ομάδες.

Το διάβασμα των πακέτων

Σε κάθε επανάληψη του βρόγχου κάθε πακέτο διαβάζεται από την global μνήμη της κάρτας σε καταχωρητές με μια εντολή. Ο λόγος είναι ότι η κεφαλίδα του πακέτου περιέχει λίγα πεδία. Αν είχε περισσότερα θα έπρεπε να αποθηκευτεί στην local μνήμη της κάρτας. Αφού διαβαστεί ένα πακέτο, επιλέγεται ο πίνακας των ομάδων με τις οποίες θα συγκριθεί, ανάλογα με το πρωτόκολλο του πακέτου. Να σημειωθεί ότι η διαδικασία αυτή γίνεται για όλα τα νήματα,

οπότε δεν υπάρχουν αποκλίνοντα νήματα στη επιλογή του πίνακα με τον οποίο θα συγκριθεί το πακέτο.

Σύγκριση διευθύνσεων IP

Ο τρόπος που γίνεται η σύγκριση της διεύθυνσης IP (αποστολής ή λήψης) του πακέτου με την διεύθυνσης IP (αποστολής ή λήψης αντίστοιχα) μιας ομάδας είναι ο εξής:

```
srcIpMatch=
(
  packet.source_ip & srcMask[tid] == srcIp[tid]
) ^ invert[tid];
```

Όπως φαίνεται και στο προηγούμενο κομμάτι κώδικα, η μάσκα δικτύου της ομάδας εφαρμόζεται στην διεύθυνση IP του πακέτου. Στη συνέχεια, τα δύο αποτελέσματα συγκρίνονται μεταξύ τους και το αποτέλεσμα της σύγκρισης γίνεται xor με το bit της αντιστροφής. Το τελικό αποτέλεσμα της πράξης είναι 0 ή 1. Ο τελεστής της αντιστροφής χρειάζεται στην περίπτωση που η διεύθυνση IP της ομάδας ορίζεται ως !147.27.1.0/24. Στη περίπτωση αυτή, στην διεύθυνση αυτή ταιριάζουν όλες οι διευθύνσεις εκτός από τις 147.27.1.0/24. Να διευκρινιστεί ότι, το bit invert έχει εξαχθεί σύμφωνα με τον προηγούμενο πίνακα αλλά αγνοήθηκε για λόγους απλότητας. Επίσης, να διευκρινιστεί ότι ο κώδικας για την διεύθυνση προορισμού είναι ακριβώς ο ίδιος.

Πιο πριν αναφέρθηκε ότι έχουν υλοποιηθεί μόνο διευθύνσεις IP τάξης A, B, C και D. Βέβαια, η προηγούμενη σύγκριση υποστηρίζει και αταξικές διευθύνσεις IP. Ο λόγος είναι ότι ο κώδικας που παράγει τις μάσκες και παρεπιπτόντως τρέχει στον host, παράγει μάσκες μόνο για CIDR μπλοκ 8, 16, 24 και 32. Δεν είναι βέβαια δύσκολο να επεκταθεί και για τυχαία CIDR μπλοκ.

Σύγκριση πορτών

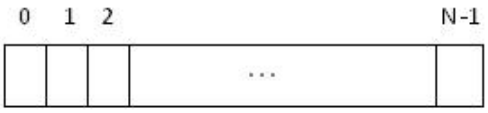
Ο τρόπος που γίνεται η σύγκριση της πόρτας (αποστολής ή λήψης) του πακέτου με το εύρος πόρτας (αποστολής ή λήψης) της ομάδας φαίνεται στο ακόλουθο κομμάτι κώδικα

```
 source_InRange=
(
  packet.source_port >=srcLow[tid] &
 packet.source_port <=srcHigh[tid]
)^invert[tid];
```

Η πόρτα συγκρίνεται αν είναι ανάμεσα στα όρια και το αποτέλεσμα γίνεται πάλι xor με το bit της αντιστροφής. Το αποτέλεσμα της σύγκρισης είναι πάλι 0 ή 1. Να διευκρινιστεί ότι το bit invert έχει εξαχθεί σύμφωνα με τον προηγούμενο πίνακα αλλά αγνοήθηκε για λόγους απλότητας. Επίσης, ο κώδικας για την πόρτα προορισμού είναι ακριβώς ο ίδιος.

Η εγγραφή των αποτελεσμάτων

Στη συνέχεια, το kernel γράφει τα αποτελέσματα σε τέσσερις πίνακες στην global μνήμη της κάρτας, ένα για κάθε πρωτόκολλο. Ο λόγος που χρησιμοποιούνται τέσσερις πίνακες, και όχι ένας, είναι ότι ο στόχος ήταν τα αποτελέσματα να γράφονται στη μορφή *το i πακέτο ταιριάζει με αυτές τις ομάδες*, όπως φαίνεται και στο σχήμα 4.8 , όπου N είναι το πλήθος των ομάδων και

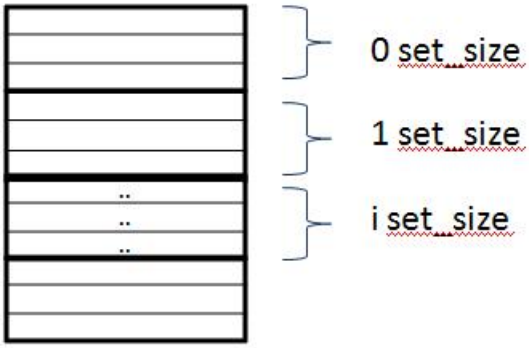


Σχήμα 4.8: Πίνακας αποτελεσμάτων για ένα πακέτο

κάθε θέση του πίνακα δηλώνει αν το πακέτο ταιριάζει με την αντίστοιχη ομάδα. Επειδή το πλήθος των ομάδων είναι διαφορετικό για κάθε πρωτόκολλο, χρησιμοποιήθηκε ένας πίνακας για κάθε πρωτόκολλο.

Το αποτέλεσμα για κάθε σύγκριση είναι μηδέν ή ένα, ανάλογα με το αν το πακέτο ταιριάζει ή όχι με την ομάδα. Να σημειωθεί ότι τα μερικά αποτελέσματα κάθε σύγκρισης αποθηκεύονται σε καταχωρητές για κάθε νήμα. Για παράδειγμα, κάθε νήμα αποθηκεύει τις μεταβλητές srcIPMatch, dstIPMatch, source_InRange και dest_InRange σε τέσσερις καταχωρητές. Το αποτέλεσμα που γράφεται στην μνήμη των αποτελεσμάτων προκύπτει σαν το λογικό ΚΑΙ των επιμέρους αποτελεσμάτων (srcIPMatch, dstIPMatch, source_InRange και dest_InRange). Ο πίνακας των αποτελεσμάτων για κάθε πρωτόκολλο είναι ένας μονοδιάστατος πίνακας N*M θέσεων, όπου N το πλήθος των ομάδων και M το πλήθος των πακέτων. Ο τρόπος που γράφονται τα αποτελέσματα είναι ο εξής:

```
result[idx + i*set_size]=
    srcIPMatch & dstIPMatch & source_InRange & dest_InRange;
```



Σχήμα 4.9: Εγγραφή αποτελεσμάτων

Ο πίνακας δηλαδή,όπως φαίνεται και στο σχήμα 4.9 χωρίζεται σε M κομμάτια και κάθε τέτοιο κομμάτι έχει μέγεθος όσο είναι και το πλήθος των ομάδων. Η θέση στην οποία θα γραφεί ένα αποτέλεσμα εξαρτάται από το νήμα που θα κάνει την εγγραφή αλλά και από το πακέτο. Για παράδειγμα το νήμα με idx 80, γράφει το αποτέλεσμα στην θέση 80 του πίνακα για το πρώτο πακέτο και στη θέση $80 + 1 * set\_size$ για το δεύτερο πακέτο.

4.5 Αντιγραφή των αποτελεσμάτων στη μνήμη του επεξεργαστή

Όταν γραφούν τα αποτελέσματα στην global μνήμη, η εκτέλεση του kernel τερματίζεται. Τότε ο host αντιγράφει τους πίνακες των αποτελεσμάτων από την device μνήμη στην host μνήμη. Η διαδικασία είναι όμοια με την αντιγραφή των ομάδων και των πακέτων στην μνήμη της κάρτας γραφικών και γίνεται με κλήση της συνάρτησης cudaMemcpy() για κάθε ένα από τους τέσσερις πίνακες των αποτελεσμάτων.

Κεφάλαιο 5

Πειραματικές μετρήσεις και σύγκριση με το Snort

Το κεφάλαιο αυτό χωρίζεται σε δύο μέρη. Αρχικά περιγράφονται τα πειράματα που διενεργήθηκαν και είχαν ως στόχο την περαιτέρω διερεύνηση και αξιολόγηση. Η διευρεύνηση αποσκοπεί στην βαθύτερη κατανόηση του μοντέλου της CUDA αλλά και στην αιτιολόγηση των επιλογών που έγιναν στην υλοποίηση. Στο δεύτερο μέρος σύγκρινεται η υλοποίηση με το Snort τόσο από άποψη ορθότητας όσο και από άποψη ταχύτητας.

Για την υλοποίηση χρησιμοποιήθηκε η κάρτα γραφικών GeForce 9800 GTX της NVIDIA, η οποία υποστηρίζει το προγραμματιστικό μοντέλο της CUDA. Τα χαρακτηριστικά του υπολογιστή που χρησιμοποιήθηκε είναι: Intel Core 2 duo @3GHz, Ram 2GB και λειτουργικό σύστημα Ubuntu Linux 8.10. Η έκδοση της CUDA που χρησιμοποιήθηκε είναι η 2.2 και τα χαρακτηριστικά της κάρτας αυτής φαίνονται στον πίνακα 5.1

5.1 Διευρεύνηση της υλοποίησης

Τα πειράματα που διενεργήθηκαν, μπορούν να χωριστούν σε δυο κατηγορίες. Η πρώτη κατηγορία αφορά τον πειραματισμό ως προς το πλήθος των νημάτων ανά μπλοκ και η δεύτερη τον προσδιορισμό του κόστους δημιουργίας νημάτων.

Πλήθος νημάτων ανά μπλοκ

Προκειμένου να διερυνηθεί ο ρόλος του πλήθους των νημάτων ανά μπλοκ, τρέξαμε ένα κενό kernel για τρία διαφορετικά μεγέθη ανά μπλοκ (32, 64 και 128 αντίστοιχα, νήματα ανά μπλοκ) για δεδομένο αριθμό νημάτων συνολικά και μετρήσαμε το χρόνο εκτέλεσης του kernel. Η μέτρηση αυτή επαναλήφθηκε

Πίνακας 5.1: Χαρακτηριστικά κάρτας γραφικών που χρησιμοποιήθηκε

CUDA Capability Major revision number	1
CUDA Capability Minor revision number	1
Total amount of global memory	536150016 byte
Number of multiprocessors	16
Number of cores	128
Total amount of constant memory	65536 bytes
Total amount of shared memory per block	16384 bytes
Total number of registers available per block	8192
Warp size	32
Maximum number of threads per block	512
Maximum sizes of each dimension of a block	512 x 512 x 64
Maximum sizes of each dimension of a grid	65535 x 65535 x 1
Maximum memory pitch	262144 bytes
Texture alignment	256 bytes
Clock rate	1.67 GHz

για διαφορετικό αριθμό νημάτων συνολικά και φαίνεται στο σχήμα 5.1. Όπως φαίνεται στο σχήμα, όσο αυξάνει ο αριθμός των νημάτων ανά μπλοκ αυξάνει ελαφρά ο χρόνος εκτέλεσης του kernel. Αυτό οφείλεται στο ότι όσο μεγαλύτερο είναι το μέγεθος ενός μπλοκ τόσο περισσότερα warps πρέπει να εκτελέσει ο πολυεπεξεργαστής.

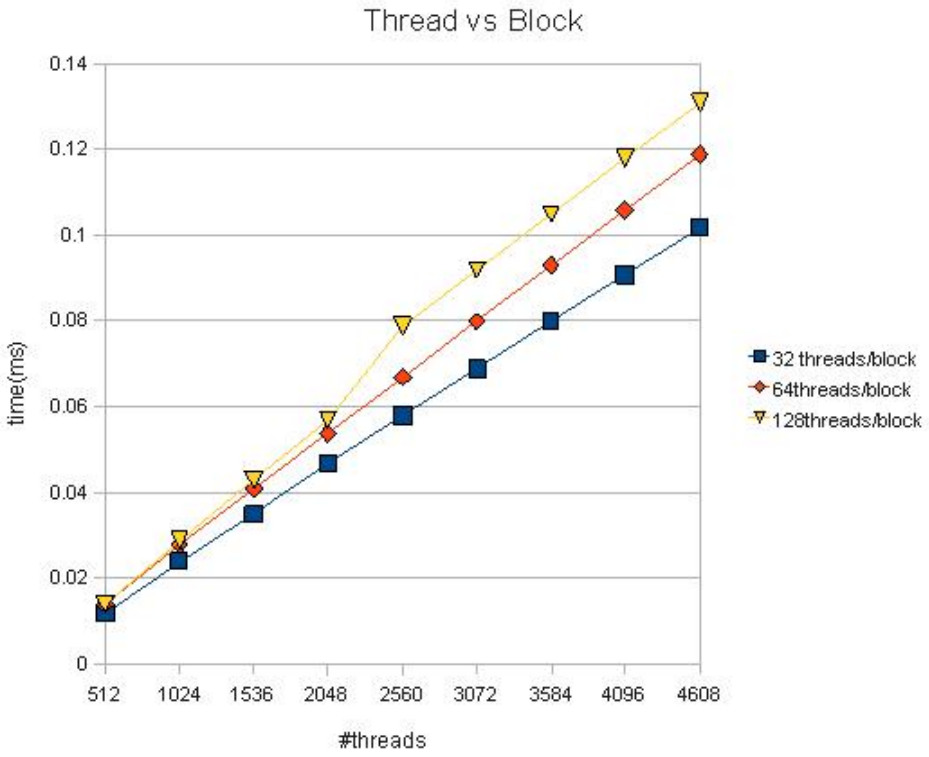
Σε δεύτερη φάση τρέξαμε το kernel της υλοποίησης (κεφάλαιο 4) για δυο διαφορετικά μεγέθη ανά μπλοκ (32 και 64 νήματα ανά μπλοκ) για δεδομένο αριθμό νημάτων συνολικά και μετρήσαμε το χρόνο εκτέλεσης. Όπως φαίνεται και στο σχήμα 5.2 ο χρόνος εκτέλεσης δεν επηρεάζεται σημαντικά από το μέγεθος των μπλοκ.

Προσδιορισμός κόστους δημιουργίας νημάτων

Στο πείραμα αυτό προσδιορίστηκε το κόστος δημιουργίας νημάτων με τον εξής τρόπο:

ένα kernel εκτελέστηκε για σύγκριση 1000 πακέτων, με δύο διαφορετικούς τρόπους. Στον πρώτο τρόπο, τα πακέτα διοχέτευτηκαν στο kernel σε ομάδες των N και η κλήση του kernel γινόταν σε ένα βρόγχο. Το μέγεθος της ομάδας είναι μεταβλητή παράμετρος και η μέτρηση επαναλήφθηκε για διαφορεικά μεγέθη ομάδας. Επομένως, ο βρόγχος εκτελέστηκε $\frac{1000}{μγεθοςομδας}$ φορές.

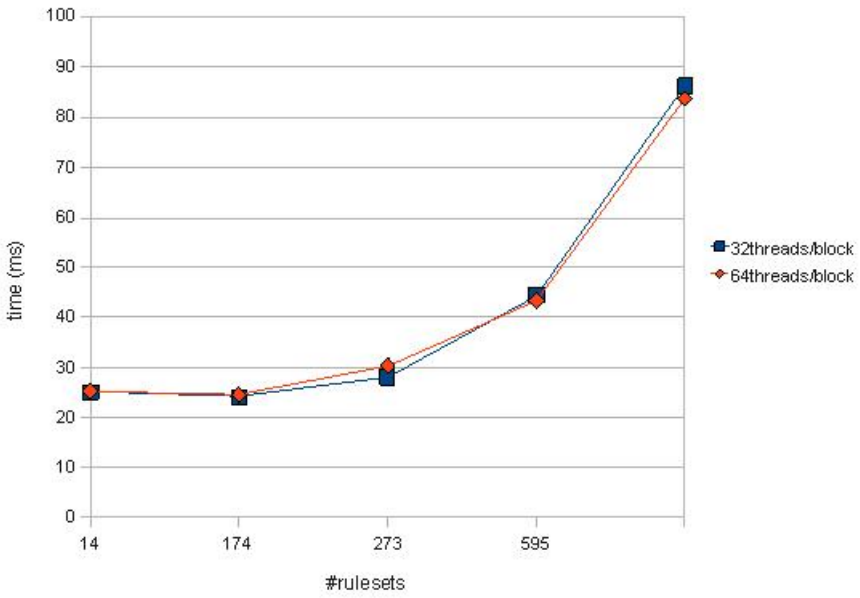
Στο δεύτερο τρόπο, το kernel λειτουργούσε όπως περιγράφηκε στο κεφά-



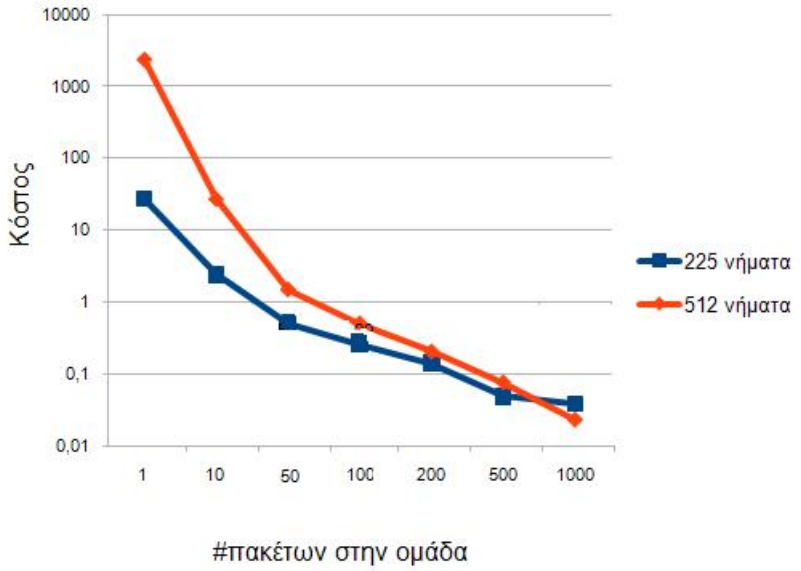
Σχήμα 5.1: Σύγκριση μεγέθους μπλοκ σε κενό kernel

λαιο 4. Η κλήση του kernel, δηλαδή, γινόταν μόνο μια φορά και το kernel διάβαζε όλα τα πακέτα μέσα σε ένα βρόγχο 1000 φορές. Ο χρόνος εκτέλεσης του πρώτου τρόπου ήταν μεγαλύτερος του δεύτερου τρόπου και η διαφορά τους είναι το κόστος δήμιουργίας νημάτων. Όπως φαίνεται και στο σχήμα 5.3, το κόστος ελαττώνεται όσο αυξάνει το μέγεθος της ομάδας. Με άλλα λόγια, είναι προτιμότερο τα νήματα να εκτελούνται όσο το δυνατόν συνεχόμενα, από το να εκτελούνται, να τερματίζουν και να ξαναξεκινούν.

Η παραπάνω μέτρηση πραγματοποιήθηκε και για μικρότερο αριθμό νημάτων και φαίνεται στο ίδιο σχήμα.



Σχήμα 5.2: Σύγκριση μεγέθους μπλοκ σε kernel υλοποίησης



Σχήμα 5.3: Καμπύλη κόστους δημιουργίας νημάτων

5.2 Σύγκριση με το Snort

Στο κομμάτι αυτό περιγράφεται ο τρόπος με το οποίο συγκρίθηκε η υλοποίηση με το Snort. Η υλοποίηση συγκρίθηκε με το Snort και από άποψη ορθότητας των αποτελεσμάτων αλλά και από άποψη απόδοσης.

5.2.1 Σύγκριση ορθότητας

Στο στάδιο αυτό πραγματοποιήθηκε η σύγκριση των αποτελεσμάτων της υλοποίησης με τα αποτελέσματα του Snort με σκοπό να επαληθευτεί η ορθότητα του προγράμματος σε CUDA. Η διαδικασία που ακολουθήθηκε για τη πραγματοποίηση της σύγκρισης μπορεί να διαχωριστεί στα παρακάτω στάδια:

1. Ρύθμιση του Snort να μην εκτελεί αναζήτηση στα περιεχόμενα του πακέτου αλλά και λοιπές ρυθμίσεις
2. Τροφοδότηση και του Snort αλλά και του παράλληλου προγράμματος με το ίδιο trace πακέτων
3. Σύγκριση των αποτελεσμάτων

Ρύθμιση του Snort

Το πρώτο στάδιο είναι απαραίτητο, επειδή το Snort εκτελεί, εκτός από την κατηγοριοποίηση των πακέτων, και αναζήτηση στα περιεχόμενα του πακέτου. Γι'αυτό το λόγο, όλοι οι κανόνες του Snort τροποποιήθηκαν, ώστε να μην εκτελούν καμία αναζήτηση στα περιεχόμενα των πακέτων. Αυτή η τροποποίηση πραγματοποιήθηκε κρατώντας την κεφαλίδα ενός κανόνα αυτούσια, αλλά από τις ρυθμίσεις του κανόνα κρατήθηκαν μόνο τα πεδία `msg, sid` και `rev`. Για παράδειγμα, ο κανόνας 1 μετατράπηκε στον κανόνα 2, όπως φαίνεται παρακάτω.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any \
(msg:"ATTACK-RESPONSES directory listing";\
flow:established; content:"Volume Serial Number";\
 classtype:bad-unknown; sid:1292; rev:9;)   (1)

alert tcp $HOME_NET any -> $EXTERNAL_NET any \
(msg:"ATTACK-RESPONSES directory listing"; sid:1292; rev:9;) (2)
```

Με αυτό το τρόπο όταν ενεργοποιηθεί ο κανόνας 2, τότε το Snort γράφει σε ένα αρχείο καταγραφής(log) το msg και τα sid και rev του κανόνα που ενεργοποιήθηκε.

Οι λοιπές ρυθμίσεις αφορούσαν την απενεργοποίηση όλων των προεπεξεργαστών και τον έλεγχο στο cheksum των πακέτων, οι οποίες πραγματοποιήθηκαν από το configuration file του Snort.

Δημιουργία κοινού trace πακέτων

Για να είναι έγκυρα τα αποτελέσματα της σύγκρισης, τα δύο προγράμματα πρέπει να έχουν την ίδια είσοδο. Το Snort μπορεί να διαβάσει αλλά και να αποθηκεύσει τα πακέτα σε ένα δυαδικό αρχείο [14], το οποίο είναι σε μορφή pcap[2] (packet capture). Το πρόβλημα που προέκυψε είναι ότι το παράλληλο πρόγραμμα χρησιμοποιεί μια απλουστευμένη μοντελοποίηση της επικεφαλίδας των πακέτων, ενώ στα pcap αρχεία, τα πακέτα είναι πραγματικά πακέτα δικτύου και οι κεφαλίδες των πακέτων περιέχουν όλα τα πεδία που πρέπει να περιέχει μια κεφαλίδα. Για παράδειγμα, μια κεφαλίδα ενός TCP πακέτου δεν περιέχει μόνο τις πόρτες προορισμού και λήψης αλλά και αριθμούς ακολουθίας, γνωστοποίησης, άθροισμα ελέγχου κ.α. Για το λόγο αυτό δημιουργήθηκε ένα pcap αρχείο μέσω του Snort (έστω το test.pcap), το οποίο στη συνέχεια με τη βοήθεια του προγράμματος Wireshark[3] μετατράπηκε σε αρχείο κειμένου (έστω το test.txt). Το παραγόμενο αυτό αρχείο κειμένου περιέχει για κάθε πακέτο μια σύντομη περιγραφή της επικεφαλίδας του με τα κυριότερα χαρακτηριστικά της, όπως για παράδειγμα για το επίπεδο δικτύου τις διευθύνσεις IP αποστολής και λήψης. Από αυτό το αρχείο κειμένου προκύπτει η είσοδος του παράλληλου προγράμματος, κρατώντας για κάθε κεφαλίδα τα πεδία που ενδιαφέρουν. Καθώς τα πεδία αυτά είναι συμβολοσειρές (strings) μετατρέπονται σε αριθμητικά δεδομένα και στη συνέχεια τροφοδοτούνται στη κάρτα γραφικών.

Σύγκριση των αποτελεσμάτων

Το επόμενο βήμα είναι η εκτέλεση των δύο προγραμμάτων, το μεν Snort με το αρχείο test.pcap σαν είσοδο και το δε παράλληλο με το test.txt. Το Snort γράφει τα αποτελέσματά του σε ένα αρχείο καταγραφής στο δίσκο. Τα αποτελέσματα είναι της μορφής

```
[sid:rev] msg PROTOCOL SourceIp:SourcePort -> DestIP:DestPort
```

όπου η περιγραφή του κάθε πεδίου φαίνεται στον πίνακα 5.2

Πιο απλά το Snort γράφει τα αποτελέσματα σε μορφή που δείχνει ποιο πακέτο ενεργοποιήσε ποιο κανόνα.

Όπως έχει περιγραφεί και στο κεφάλαιο 4 τα αποτελέσματα από το παράλληλο πρόγραμμα αποθηκεύονται σε ένα πίνακα από μηδέν και ένα, όπου το

Πίνακας 5.2: Περιγραφή των πεδίων καταγραφής του Snort

sid	Το sid του κανόνα που ενεργοποιήθηκε
rev	Το πεδίο rev του κανόνα που ενεργοποιήθηκε
msg	Το μήνυμα του κανόνα που ενεργοποιήθηκε
PROTOCOL	Το πρωτόκολλο του πακέτου που ενεργοποίησε τον κανόνα
SourceIp	Η διεύθυνση IP αποστολής του πακέτου που ενεργοποίησε το κανόνα
SourcePort	Η πόρτα αποστολής του πακέτου που ενεργοποίησε το κανόνα
DestIp	Η διεύθυνση IP λήψης του πακέτου που ενεργοποίησε το κανόνα
DestPort	Η πόρτα λήψης του πακέτου που ενεργοποίησε το κανόνα

μηδέν δηλώνει ότι ένα πακέτο δεν ταιριάζει στην ομάδα ενώ το ένα, ότι το πακέτο ταιριάζει στην ομάδα. Ο πίνακας αυτός, όταν τελειώσει η εκτέλεση του kernel, αντιγράφεται στη κύρια μνήμη του host για περαιτέρω επεξεργασία. Όσον αφορά στη σύγκριση ορθότητας, η περαιτέρω αυτή επεξεργασία αφορά στη μετατροπή αυτού του πίνακα σε μια μορφή συγκρίσιμη με την έξοδο του Snort. Το Snort γράφει στην έξοδο κανόνες που ενεργοποιήθηκαν, ενώ το παράλληλο πρόγραμμα ομάδες κανόνων. Γι'αυτό το λόγο, από το πίνακα των αποτελεσμάτων του παράλληλου προγράμματος όταν βρίσκουμε ένα, γράφουμε σε ένα αρχείο κειμένου τους κανόνες (στη πραγματικότητα μόνο τα πεδία sid και rev του κανόνα) της ομάδας που ενεργοποιήθηκε. Με αυτό το τρόπο, η έξοδος του παράλληλου προγράμματος έχει συγκρίσιμη μορφή με την έξοδο του Snort. Συγκρίνοντας τα δύο αρχεία προέκυψε ότι τα αποτελέσματα του παράλληλου προγράμματος είναι υπερσύνολο των αποτελεσμάτων του Snort. Το γεγονός αυτό θα επεξηγηθεί με το παρακάτω παράδειγμα.

Μετά την εκτέλεση του παράλληλου προγράμματος ενεργοποιήθηκαν οι

ομάδες κανόνων που φαίνονται στον πίνακα 5.3, οι οποίοι αναφέρονται στο πρωτόκολλο UDP. Τα πακέτα που δώθηκαν που ταίριαζαν στις παραπάνω ομά-

Πίνακας 5.3: Παραγόμενες ομάδες κανόνων

Ομάδα	**IP αποστολής**	**Πόρτα αποστολής**	**IP λήψης**	**Πόρτα λήψης**
20	$HOME_NET	any	$EXTERNAL_NET	any
34	any	any	any	53
76	$HOME_NET	any	$EXTERNAL_NET	53
21	$EXTERNAL_NET	any	$HOME_NET	any
33	$EXTERNAL_NET	53	$HOME_NET	any
66	$EXTERNAL_NET	any	$HOME_NET	1024:
80	$EXTERNAL_NET	any	$HOME_NET	500:

δες είχαν τις παραμέτρους που φαίνονται στο πίνακα 5.4

Πίνακας 5.4: Παράμετροι πακέτων

IP αποστολής	Πόρτα αποστολής	IP λήψης	Πόρτα λήψης
HOME_NET	35657	EXTERNAL_NET	53
EXTERNAL_NET	53	HOME_NET	35657

Η πρώτη στήλη του πίνακα 5.3 αναφέρεται στον αριθμό της ομάδας που ενεργοποιήθηκε και αντιστοιχεί στη θέση της ομάδας μέσα στον πίνακα των ομάδων. Οι υπόλοιπες στήλες αναφέρονται στις παραμέτρους των ομάδων. Ενώ λοιπόν, σύμφωνα με το παράλληλο πρόγραμμα ενεργοποιούνται οι ομάδες 20, 34, 76, 21, 33, 66 και 80, μετά την εκτέλεση του Snort ενεργοποιήθηκαν οι κανόνες που αντιστοιχούν στις ομάδες 34, 76 και 33. Ο λόγος για τον οποίο γίνεται αυτό είναι επειδή το Snort, όταν δύο κανόνες ενεργοποιούνται για ένα πακέτο διαλέγει να καταγράψει αυτόν με την πιο μοναδική παράμετρο[4]. Στη περίπτωση που το πρωτόκολλο είναι TCP ή UDP, καταγράφεται ο κανόνας με την πιο μοναδική πόρτα. Όπως φαίνεται και από τους προηγούμενους πίνακες, τα πακέτα στη πρώτη γραμμή του πίνακα 5.4 ταιριάζουν με τις τρεις πρώτες ομάδες του πίνακα 5.3. Παρόλα αυτά, το Snort επιλέγει τις ομάδες 34 και 76 επειδή στην πόρτα προορισμού έχουν συγκεκριμένη πόρτα, την 53.

Αντίστοιχα, τα πακέτα της δεύτερης γραμμής του πίνακα 5.4 ταιριάζουν με τις τρεις τελευταίες ομάδες του πίνακα 5.3, αλλά το Snort καταγράφει μόνο την ομάδα 33.

Επειδή τα αποτελέσματα του παράλληλου προγράμματος είναι υπερσύνολο των αποτελεσμάτων του Snort, ο τρόπος με τον οποίο κάποιος θα μπορεί να πάει από τα αποτελέσματα του παράλληλου προγράμματος στα αποτελέσματα του Snort είναι να βρεθεί η πιο μοναδική πόρτα. Για κάθε ομάδα που ενεργοποιήθηκε από το παράλλήλο πρόγραμμα, υπολογίζεται η διαφορά ανάμεσα στο άνω και στο κάτω όριο της πόρτας. Πιο μοναδική θεωρείται αυτή για την οποία η διαφορά είναι πιο κοντά στο μηδέν. Αν πρόκειται για απλή πόρτα, τότε το άνω και το κάτω όριο είναι ίδια και η διαφορά τους μηδέν. Αν πρόκειται για εύρος πόρτας, τότε πιο μοναδική είναι αυτή με τη μικρότερη διαφορά.

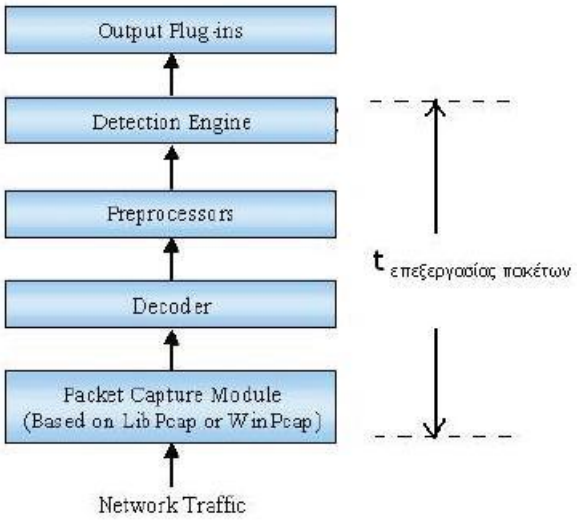
5.2.2 Σύγκριση απόδοσης

Το επόμενο βήμα μετά τη σύγκριση ορθότητας ήταν η σύγκριση όσον αφορά στην απόδοση των δυο προγραμμάτων. Για το λόγο αυτό μετρήθηκε ο χρόνος εκτέλεσης του Snort και ο χρόνος εκτέλεσης του παράλληλου προγράμματος. Επειδή όμως τα δύο προγράμματα δεν εκτελούν ακριβώς τις ίδιες λειτουργίες ο συνολικός χρόνος εκτέλεσης των δυο προγραμμάτων δεν είναι κατάλληλος για την εξαγωγή συμπερασμάτων. Για αυτό το λόγο μετρήθηκε ο χρόνος ανίχνευσης, ο οποίος είναι ο χρόνος που χρειάζεται για την εφαρμογή των κανόνων στα πακέτα. Ο χρόνος δηλαδή, για να συγκριθούν τα πακέτα με τους κανόνες και να αποφασιστεί ποιο πακέτο ταιριάζει σε ποιον κανόνα.

Μέτρηση Χρόνου Ανίχνευσης του Snort

Σε κάθε εκτέλεση του Snort, το ίδιο το Snort αναφέρει στον χρήστη το χρόνο επεξεργασίας πακέτων. Αυτός ο χρόνος είναι ο χρόνος που χρειάζεται το Snort για να αποφασίσει αν ένα σύνολο πακέτων ταιριάζει σε ένα σύνολο κανόνων. Όπως φαίνεται και στο σχήμα 5.4 ο χρόνος αυτός περιλαμβάνει το χρόνο ανάγνωσης των πακέτων από το pcap αρχείο, το χρόνο αποκωδικοποίησης των πακέτων, το χρόνο να περάσει από τους προεπεξεργαστές και το χρόνο που ξοδεύεται στη μηχανή ανίχνευσης του Snort. Αυτός ο χρόνος δεν είναι ακριβώς ο ζητούμενος χρόνος, δηλαδή ο χρόνος ανίχνευσης. Από τη μελέτη του πηγαίου κώδικα του Snort προέκυψε ότι ο χρόνος επεξεργασίας πακέτων που αναφέρει το Snort, προκύπτει ως εξής:

Το Snort διαβάζει κάθε πακέτο από το pcap αρχείο με χρήση της συνάρτησης pcap_dispatch της βιβλιοθήκης pcap [2]. Η συνάρτηση pcap_dispatch δέχεται σαν όρισμα το όνομα μιας συνάρτησης, η οποία εκτελείται κάθε φορά που διαβάζεται ένα πακέτο. Η συνάρτηση αυτή στην ορολογία της βιβλιοθήκης



Σχήμα 5.4: Ο χρόνος επεξεργασίας πακέτων που επιστρέφει το Snort

ονομάζεται callback συνάρτηση. Στη περίπτωση του Snort σαν callback έχει οριστεί η συνάρτηση που επεξεργάζεται το πακέτο. Ο χρόνος που αναφέρει το Snort προκύπτει ως:

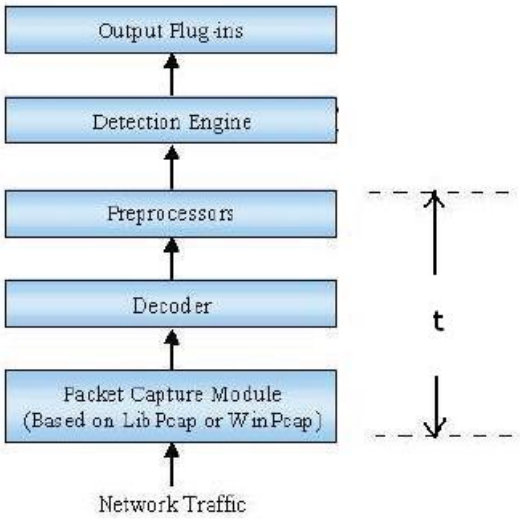
```
gettimeofday(starttime);
pcap_dispatch(...,callback);
gettimeofday(endtime);
```

Στην ουσία ο χρόνος αυτός είναι ο χρόνος που χρειάζεται η συνάρτηση callback για να εκτελεστεί. Για να μετρηθεί ο χρόνος ανίχνευσης πραγματοποιήθηκε η παρακάτω διαδικασία:

Η συνάρτηση callback τροποποιήθηκε ώστε να μην κάλει τη μηχανή ανίχνευσης. Ο χρόνος τότε που προκύπτει αν εκτελεστεί το Snort περιλαμβάνει το χρόνο ανάγνωσης των πακέτων από το pcap αρχείο, το χρόνο αποκωδικοποίησης των πακέτων και το χρόνο να περάσει από τους προεπεξεργαστές, όπως φαίνεται και στο σχήμα 5.5 . Η διαφορά των δύο χρόνων είναι ο χρόνος που ξοδεύεται στη μηχανή ανίχνευσης, που είναι και το ζητούμενο.

Μέτρηση Χρόνου Ανίχνευσης στο παράλληλο πρόγραμμα

Η μέτρηση του χρόνου ανίχνευσης στο παράλληλο πρόγραμμα προκύπτει αν μετρήσουμε το χρόνο εκτέλεσης του kernel. Ο χρόνος αυτός αντιστοιχεί στο



Σχήμα 5.5: Ο χρόνος επεξεργασίας χωρίς τη μηχανή ανίχνευσης

χρόνο που χρειάζεται να συγκριθούν τα πακέτα με τις ομάδες και στην εγγραφή των αποτελεσμάτων. Να σημειωθεί ότι, οι ομάδες και τα πακέτα βρίσκονται ήδη στη global μνήμη της κάρτας γραφικών, οπότε δεν συμπεριλαμβάνεται στη μέτρηση ο χρόνος αντιγραφής των πακέτων και των κανόνων στη κάρτα. Η μέτρηση αυτή έγινε με χρήση χρονομετρητών κατάλληλων για την κάρτα γραφικών και είναι διαθέσιμοι από το API της CUDA μέσω της συνάρτησης CudaEvent.

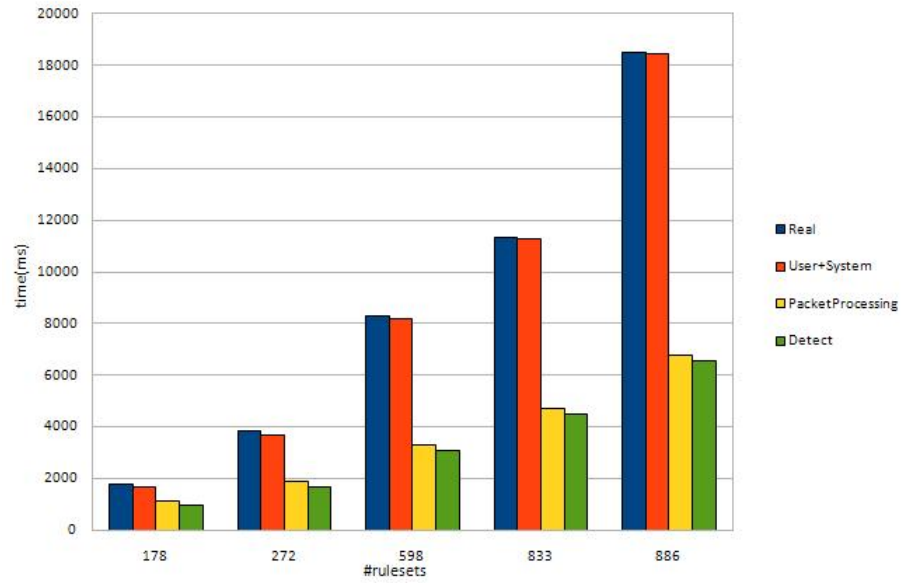
Εκτός από το χρόνο ανίχνευσης μετρήθηκαν επίσης:

- ο χρόνος που χρειάζεται για την ομαδοποιήση των κανόνων
- ο συνολικός χρόνος που ξοδεύεται στις αντιγραφές:
  - ομάδων από το host στο device
  - πακέτων από το host στο device
  - αποτελεσμάτων από το device στο host

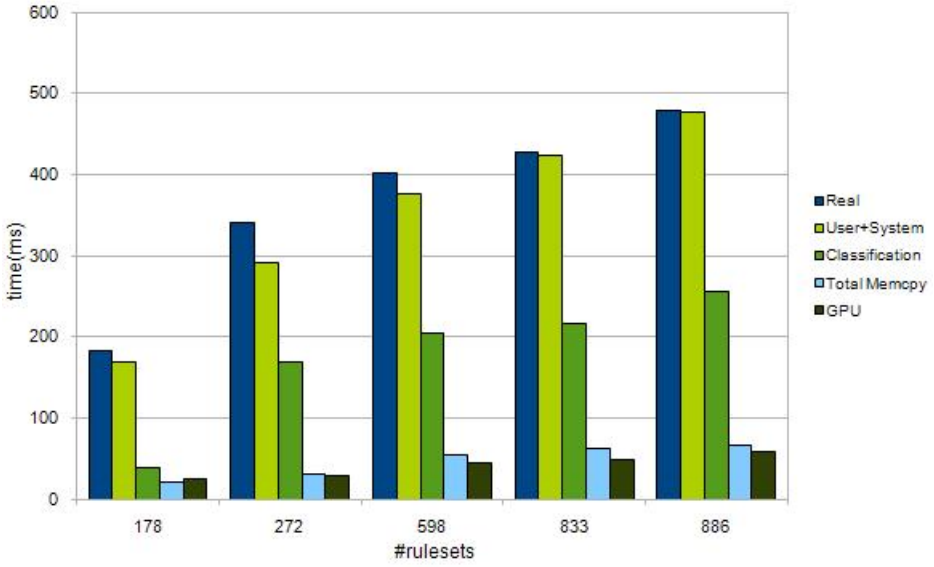
Σύγκριση Χρόνων και Απόδοσης

Προκειμένου να μετρηθεί και να συγκριθεί η απόδοση των δυο προγραμμάτων, εκτελέστηκαν με σταθερό αριθμό πακέτων και αυξανόμενο αριθμό κανόνων. Με αυτόν το τρόπο ερευνήθηκε ο τρόπος με τον οποίο μεταβάλλεται ο χρόνος εκτέλεσής τους με την αύξηση του αριθμού των κανόνων. Ο αριθμός των πακέτων ήταν 22.320. Και στα δυο προγράμματα εκτός από το χρόνο ανίχνευσης,

όπως περιγράφηκε παραπάνω, μετρήθηκε και ο συνολικός χρόνος εκτέλεσης με χρήση της εντολής time στο linux. Οι χρόνοι που μετρήθηκαν φαίνονται στο σχήμα 5.6



(a) snort



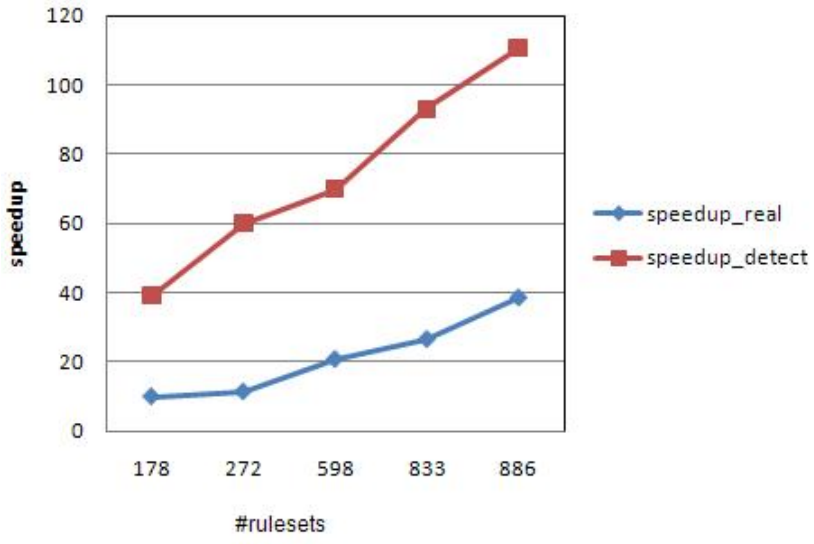
(b) cuda

Σχήμα 5.6: Χρόνοι εκτέλεσης στο Snort και στο παράλληλο πρόγραμμα

Στο σχήμα 5.6(a), ο χρόνος Real αναφέρεται στο χρόνο real που επιστρέφεται από τη εντολή time. Ο χρόνος User+System, αναφέρεται στο άθροισμα των χρόνων user και system που επιστρέφονται από την time. Ο χρόνος PacketProcessing είναι ο χρόνος που αναφέρει το snort όταν εκτελείται και αντιστοιχεί στο χρόνο που φαίνεται στο σχήμα 5.4. Ο χρόνος Detect είναι ο χρόνος που προκύπτει, όπως αναφέρεται στη σελίδα 70.

Αντίστοιχα, στο σχήμα 5.6(b), οι χρόνοι Real, User και System είναι οι χρόνοι όπως προκύπτουν από την time. Ο χρόνος Classification είναι ο χρόνος που χρειάζεται ο host για να ομαδοποιήσει τους κανόνες και να τους τοποθετήσει στους τέσσερις πίνακες, όπως αναφέρθηκε στο κεφάλαιο refimplementation. Ο χρόνος TotalMemcpy είναι ο χρόνος που ξοδεύεται σε όλες τις αντιγραφές ομάδων, πακέτων και αποτελεσμάτων από και προς την κάρτα γραφικών. Ο χρόνος GPU είναι ο χρόνος εκτέλεσης του kernel, όπως αναφέρθηκε στη σελίδα 71.

Όπως φαίνεται και στο σχήμα 5.6, ο χρόνος ανίχνευσης στο παράλληλο πρόγραμμα είναι σχεδόν μια τάξη μεγέθους μικρότερος από το χρόνο ανίχνευσης στο Snort. Η επιτάχυνση που επιτυγχάνεται υπολογίζεται διαιρώντας το χρόνο ανίχνευσης του Snort με το χρόνο ανίχνευσης του παράλληλου προγράμματος. Αντίστοιχα, υπολογίστηκε και η επιτάχυνση όσον αφορά στον πραγματικό χρόνο εκτέλεσης του προγράμματος, αλλά όπως αναφέρθηκε παραπάνω ο συνολικός χρόνος εκτέλεσης των δυο προγραμμάτων δεν είναι αξιόπιστο μέτρο σύγκρισης. Όπως φαίνεται και στο σχήμα 5.7 η επιτάχυνση της



Σχήμα 5.7: Επιτάχυνση της απόδοσης

απόδοσης που επιτυγχάνεται μέσω του παράλληλου προγράμματος, κυμαίνεται από 42x έως 109x ανάλογα με το πλήθος των ομάδων, όσον αφορά στο χρόνο ανίχνευσης και από 4x έως 40x για τον συνολικό χρόνο εκτέλεσης.

Κεφάλαιο 6

Συμπεράσματα - Μελλοντικές επεκτάσεις

6.1 Συμπεράσματα

Η παρούσα διπλωματική ασχολήθηκε με την υλοποίηση της κατηγοριοποίησης της κεφαλίδας πακέτων δικτύου με παράλληλο τρόπο. Γι' αυτό το λόγο, χρησιμοποιήθηκε το παράλληλο προγραμματιστικό μοντέλο της CUDA, το οποίο χρησιμοποιείται σε όλες τις κάρτες γραφικών της Nvidia. Σαν σύστημα αναφοράς χρησιμοποιήθηκε το Snort, ένα ευρέως χρησιμοποιούμενο σύστημα ανίχνευσης εισβολών σε δίκτυα. Στόχος ήταν η κατηγοριοποίηση να είναι αποδοτικότερη από το Snort.

Όπως παρατηρήθηκε από τα αποτελέσματα, με την παράλληλη κατηγοριοποίηση μπορεί να επιτευχθεί επιτάχυνση από 40x εώς 100x, έναντι του τρόπου κατηγοριοποίησης του Snort, ανάλογα με το πλήθος των κανόνων που χρησιμοποιείται.

6.2 Μελλοντικές επεκτάσεις

Η αρχιτεκτονική που σχεδιάστηκε και υλοποιήθηκε στην εργασία αυτή, επιδέχεται αλλάγες που ενδεχομένως να οδηγήσουν σε βελτιστοποιήσεις. Αρχικά θα μπορούσε να επεκταθεί το επίπεδο παραλληλισμού που είχε υιοθετηθεί και εκτός από παραλληλία σε επίπεδο ομάδας κανόνων, να υλοποιηθεί και παραλληλία σε επίπεδο πακέτου. Δηλαδή, αντί να ανατίθεται ένα νήμα σε κάθε ομάδα, να ανατίθεται ένα νήμα σε κάθε ομάδα και κάθε πακέτο. Αυτό το επίπεδο παραλληλίας, θα μπορούσε να υλοποιηθεί με χρήση των πολυδιάστατων πλεγμάτων, μπλοκ και νημάτων που υποστηρίζει το μοντέλο της CUDA.

Επίσης, θα μπορούσε να υλοποιηθεί ένας πιο ρεαλιστικός τρόπος διοχέτευσης των πακέτων στη κάρτα γραφικών. Πιο συγκεκριμένα, αντί το σύνολο των πακέτων να διοχετεύεται εξ' ολοκλήρου στη κάρτα, θα μπορούσε να υλοποιηθεί ένα σύστημα παραγωγού-καταναλωτή, στο οποίο ο παραγωγός-host θα τοποθετεί πακέτα στην global μνήμη της κάρτας και ο καταναλωτής-device θα διαβάζει τα πακέτα και θα τα επεξεργάζεται.

Τέλος, σε μακροσκοπικό επίπεδο, η εργασία αυτή υλοποιεί το πρώτο στάδιο επεξεργασίας του Snort, που είναι η κατηγοριοποίηση των πακέτων, με παράλληλο τρόπο. Το σύστημα Gnort [26] υλοποιεί το δεύτερο στάδιο επεξεργασίας του Snort, που είναι η αναζήτηση των ύποπτων συμβολοσειρών στα περιεχόμενα των πακέτων, με παράλληλο τρόπο. Θα μπορούσε να υλοποιηθεί ένας συνδυασμός των δυο συστημάτων, έτσι ώστε και τα δυο στάδια επεξεργασίας του Snort να εκτελούνται με παράλληλο τρόπο.

Βιβλιογραφία

[1] http://developer.nvidia.com/page/cg_main.html.

[2] http://www.tcpdump.org.

[3] http://www.wireshark.org.

[4] Alert ordering. Snort Readme.

[5] A.Aho and M.Corasick. Fast pattern matching:an aid to bibliographic search. *Communn. ACM*, 18(6):333–340, June 1975.

[6] Vassilis Dimopoulos, Giorgos Papadopoulos, and Dionisios Pnevmatikatos. On the Importance of Header Classification in HW/SW network Intrusion Detection Systems. In *10th Panhellenic Conference on Informatics (PCI)*, November 11-13 2005. Volos, Greece.

[7] Vassilis Dimopoulos, Ioannis Papaefstathiou, and Dionisios Pnevmatikatos. A Memory-Efficient Reconfigurable Aho-Corasick FSM Implementation for Intrusion Detection Systems. In *International Conference on Embedded Computer Systems: Architectures, Modelling and Simulation (IC-SAMOS 2007)*, July 16-19 2007. Samos, Greece.

[8] Nigel Jacob and Carla Brodley. Offloading IDS Computation to the GPU. In *22nd Annual Computer Security Applications Conference, ACSAC '06*, 2006.

[9] NVIDIA. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, June 2008. http://www.nvidia.com.

[10] NVIDIA. *NVIDIA CUDA C Programming Best Practices Guide*, July 2009. http://www.nvidia.com.

[11] Giorgos Papadopoulos and Dionisios Pnevmatikatos. HASHING+MEMORY=LOW COST, EXACT PATTERN MATCHING. In

15th International Conference on Field Programmable Logic and Applications (FPL), August 24-26 2005. Tampere, Finland.

[12] PeakStream. http://arstechnica.com/hardware/news/2006/09/7763.ars.

[13] Dionisios Pnevmatikatos and Aggelos Arelakis. VARIABLE-LENGTH HASHING FOR EXACT PATTERN MATCHING. In *16th International Conference on Field Programmable Logic and Applications (FPL)*, August 28-30 2006. Madrid, Spain.

[14] The Snort Project. *Snort Users Manual.*

[15] RapindMind. Multicore development platform. http://www.rapindmind.com/product.php.

[16] R.Boyer and J.Moore. A fast string matching algorithm. *Commun. ACM*, 20(10):762–772, October 1977.

[17] Tom R.Halfhill. Parallel processing with cuda.nvidia's high-perfomance computing platform uses massive multithreading. Technical report, MICROPROCERROR REPORT, 01 2008.

[18] Martin Roesch. Snort -lightweight intrusion detection for networks.

[19] Ioannis Sourdis, Vasilis Dimopoulos, Dionisios Pnevmatikatos, and Stamatis Vassiliadis. Packet Pre-filtering for Network Intrusion Detection. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, December 4-5 2006. San Jose, USA.

[20] Ioannis Sourdis and Dionisios Pnevmatikatos. Fast, Large-Scale String Match for 10Gbps FPGA-based Network Intrusion Detection System. In *13th International Conference on Field Programmable Logic and Applications (FPL)*, September 1-3 2003. Lisbon.

[21] Ioannis Sourdis and Dionisios Pnevmatikatos. Pre-decoded CAMs for Efficient and High-Speed NIDS Pattern Matching. In *Twelfth Annual IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, April 20-23 2004. Napa, California, USA.

[22] Ioannis Sourdis, Dionisios Pnevmatikatos, Stephan Wong, and Stamatis Vassiliadis. A RECONFIGURABLE PERFECT-HASHING SCHEME FOR PACKET INSPECTION. In *15th International Conference on Field Programmable Logic and Applications (FPL)*, August 24-26 2005. Tampere, Finland.

[23] S.Wu and U.Manber. A fast algorithm for multi-pattern searching. Tr-94-17, University of Arizona, 1994.

[24] Tilera. http://www.tilera.com/products/processors.php.

[25] Alok Tongaonkar, Sreenaath Vasudevan, and R. Sekar. Fast Packet Classification for Snort by Native Compilation of Rules. In *22nd Large Installation System Administration Conference (LISA '08)*, 2008.

[26] Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos P. Markatos, and Sotiris Ioannidis. Gnort: High performance network intrusion detection using graphics processors.

[27] Δημόπουλος Βασίλης. Διπλωματική Εργασία, Αξιολόγηση Λογισμικού Ανίχνευσης Δικτυακών Εισβολών με Υποβοήθηση από Υλικό, 2004.